الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة

Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا

Faculté de Technologie

قسم الإلكترونيك

Département d'Électronique

Mémoire de Master

Filière : Télécommunication

Spécialité : Systèmes de Télécommunication **ST8**

Présenté par

Mr. Mohamed Nazim BOUHOUNALI

&

Mr. Cherif Ouassim BRAHMI

Conception et implémentation d'un système intelligent de détection de la somnolence à l'aide de l'apprentissage profond

Proposé Par :

Pr. Abderrahmane NAMANE.

Dr. Bilal AOUINANE.

Année Universitaire **2024-2025.**

Remerciements

*Tout d'abord, nous exprimons notre gratitude la plus profonde à **Allah**, le Tout Puissant, pour nous avoir accordé la force, la guidance et la sagesse tout au long de notre parcours de recherche et de la rédaction de notre mémoire.*

*Nous exprimons ensuite notre profonde gratitude à **Pr. NAMANE Abderrahmane**, notre promoteur, pour son encadrement et ses conseils tout au long de ce projet. Son expertise et sa confiance en notre travail ont été des moteurs essentiels pour mener à bien cette étude.*

*Nous remercions également **Dr. AOUINANE Bilel**, notre Co-Promoteur, pour son soutien et son aide. Sa disponibilité et son engagement ont grandement contribué à la qualité de ce travail.*

*Nous sommes honorés d'avoir eu l'opportunité de présenter nos recherches devant des personnalités aussi distinguées, à savoir **Mr. Ait Saadi** et **Mr. Dahmani** pour leur évaluation méticuleuse et leurs commentaires constructifs.*

Enfin, nous renouvelons nos remerciements à ceux qui nous ont aidés de près ou de loin pour réaliser ce travail sans oublier les enseignants qui ont contribués à notre formation.

Dédicaces

Avec l'expression de ma reconnaissance, je dédie ce modeste travail à ceux qui, quels que soient les termes embrassés, je n'arriverais jamais à leur exprimer mon amour sincère.

A l'homme, mon précieux offre de dieu, à qui je dois ma réussite et tout mon respect

Mon cher père **Brahim**

A la femme qui a souffert sans me laisser souffrir, qui n'a jamais dit non à mes souhaits et qui n'a épargné aucun effort pour me rendre heureux

Ma chère mère **Amina**

Je dédie cet évènement marquant de ma vie à la mémoire de mes grands-parents paternel, **Mohamed** et **Djerbi Messaouda** partis trop tôt. J'espère qu'ils apprécient cet humble geste comme preuve de reconnaissance de la part d'un fils qui a toujours prié pour le salut de leurs âmes. Puisse Dieu, les avoir en sa sainte miséricorde.

A mes grands-parents maternels, **Tafadjira Hadj Slimane** et **Yagoub Saliha**, qui n'ont cessé de formuler des prières à mon égard. Que Dieu leurs donne une longue et joyeuse vie.

A mon cher frère **Zakaria**, mes **oncles** et mes **tantes** qui n'ont pas cessé de me conseiller, encourager et soutenir tout au long de mes études. Que dieu les protège et leurs offre la santé et le bonheur.

A tous les **cousins**, les **amis** que j'ai connu jusqu'à maintenant, Merci pour leurs encouragements

Sans oublier mon binome **Cherif Ouassim Brahmi** pour son soutien moral, sa patience et sa compréhension tout au long de ce projet.

Enfin, à toutes celles et ceux qui m'ont soutenu, encouragé, conseillé, même d'un simple mot ou d'un geste,
Merci. Ce mémoire est autant le vôtre que le mien.

Mohamed Nazim BOUHOUNALI

Dédicaces

Je dédie ce mémoire avec tout mon amour, ma reconnaissance et mon respect :

À mon cher père **Redha**, un modèle de sagesse, de travail et de patience,
et à ma chère mère **Chadia**, dont les prières, la tendresse et le courage m'ont toujours porté.
Votre amour inconditionnel et vos sacrifices silencieux ont été la source de ma force dans les moments les plus difficiles.
Ce travail est l'aboutissement de vos rêves pour moi, et je vous en suis profondément reconnaissant.

À mes frères et sœurs, présents en Algérie et à l'étranger,
Merci pour votre soutien, vos messages d'encouragement et votre foi en moi, même à distance.
Votre présence m'a toujours rappelé que je ne suis jamais seul, même lorsque tout semble compliqué.

À ma professeure principale, Madame **Boutaleb**,
Votre bienveillance, votre écoute et votre conseils éclairés et votre aide précieuse tout au long de mon Master 2.
Votre implication a largement dépassé le cadre académique, et je vous en suis infiniment reconnaissant.

À toutes les personnes qui m'ont aidé de près ou de loin dans la constitution et la préparation des datasets,
Votre disponibilité, votre patience et votre générosité ont permis à ce projet de prendre forme et d'aller jusqu'au bout.
Merci d'avoir cru en ce travail et d'y avoir contribué, parfois dans l'ombre, mais toujours avec cœur.

À mon binôme et ami **Mohamed Nazim**,
Merci pour ta rigueur, ton esprit d'équipe, ta persévérance et ton amitié.
Ce mémoire, nous l'avons construit ensemble, pas à pas, avec passion et détermination.
Travailler à tes côtés a été une chance et un véritable plaisir.

Enfin, à toutes celles et ceux qui m'ont soutenu, encouragé, conseillé, même d'un simple mot ou d'un geste,
Merci. Ce mémoire est autant le vôtre que le mien.

Cherif Ouassim BRAHMI

الملخص

يهدف هذا المشروع إلى تصميم نظام للكشف والتعرف على حالة العينين باستخدام التعلم العميق وتقنيات معالجة الصور. المعروف بدقته وقدرته على اكتشاف الأجسام في الوقت الحقيقي، مع تطبيق تقنيات YOLOv11 لقد استخدمنا نماذج متقدمة في معالجة الصور لتحديد موضع العينين بدقة في الصورة والتعرف على حالتهما سواء كانتا مفتوحتين أو مغلقتين. كما قمنا بتطوير واجهة رسومية وتطبيق على الهاتف المحمول لتسهيل استخدام نظامنا.

Résumé

L'objectif de ce Projet consiste à concevoir un système qui sert à détecter et faire la reconnaissance de l'état des yeux à l'aide de l'apprentissage profond et les techniques de traitement d'image. Nous avons utilisé un modèle YOLO V11 qui est reconnu par sa précision et sa capacité à détecter les objets en temps réel tout en appliquant les techniques de traitement d'images avancées afin de reconnaitre précisément la position des yeux dans une image et identifier son état s'ils sont fermés ou bien ouverts. Nous avons aussi créé une interface graphique et une application mobile pour faciliter l'utilisation de notre système.

Abstract

The objective of this project is to design a system for detecting and recognizing the state of the eyes using deep learning and image processing techniques. We used a YOLO V11 model, known for its accuracy and ability to detect objects in real time, while applying advanced image processing techniques to precisely locate the eyes in an image and determine whether they are open or closed. We also developed a graphical user interface and a mobile application to facilitate the use of our system.

Table des matières

Liste des figures

Liste des tableaux

Listes des acronymes et abréviations

PERCLOS : Percentage of Eye Closure

CNN : Convolutional Neural Network

YOLO : You Only Look Once

ROI : Region Of Interest

MLP : Multi-Layer Perceptron

IoU : Intersection Over Union

EEG : L'électroencéphalographie

EOG : l'électro-oculographie

ECG : l'électrocardiographie

HRV : Heart Rate Variability

FPS : Frames Per Second

SAT : Self-Adversarial Training

SPP : Spatial Pyramid Pooling

API : Application Programming Interface

C2PSA : Cross Stage Partial with Spatial Attention

ReLU : Rectified Linear Unit

NMS : Non-Max Suppression

CSP : Cross Stage Partial

SPFF : Spatial Pyramid Pooling Fast

SIoU : Spatial Location Unertainty

E-ELAN : Extended Efficient Layer Aggregation Network

COCO : Common Objects in Context

ADAS : Advanced Driver Assistance Systems

GPU : Graphics Processing Unit

OBB : Oriented Bounding Box

DPI : Dots Per Inch

VS Code : Visual Studio Code

PIL : Python Imaging Library

GUI : Graphical User Interface

CPU : Central Processing Unit

MVVM : Model-View-ViewModel

SVM : Support Vector Machine

PAFPN : Probabilistic Anchor-Free Feature Pyramid Network

PANet : Path Aggregation Network

mAP : Mean Average Precision

JPEG : Joint Photographic Experts Group

ONNX : Open Neural Network Exchange

VP : Vrai Positif

FP : Faux Positif

FN : Faux Négatif

Introduction Générale

La somnolence représente un danger important dans de nombreux contextes critiques, notamment dans les domaines des transports, de l'industrie, ou encore de la surveillance. L'endormissement involontaire, même de quelques secondes, peut avoir des conséquences graves, voire fatales. Face à ce constat, il devient essentiel de concevoir des systèmes capables de détecter de manière fiable et en temps réel les signes avant-coureurs de la somnolence.

Avec les avancées récentes en intelligence artificielle et en vision par ordinateur, il est désormais possible de développer des outils non intrusifs capables d'analyser des signaux visuels, comme le clignement des yeux ou leur fermeture prolongée, pour évaluer l'état de vigilance d'un individu. Parmi les techniques les plus performantes, les modèles de détection d'objets en temps réel, tels que YOLO (You Only Look Once), se sont imposés comme des solutions efficaces, rapides et adaptées aux environnements embarqués.

Dans le cadre de ce projet, nous avons conçu et développé une application mobile intelligente capable de détecter les signes de somnolence à partir de l'analyse des yeux. Le système repose sur un pipeline complet allant de la prise d'images, à l'annotation des données via Roboflow, en passant par l'entraînement d'un modèle YOLOv11 sur Google Colab, jusqu'à l'intégration du modèle dans une application Android via TensorFlow Lite. Une interface de test locale a également été développée avec Tkinter pour valider les performances du modèle avant le déploiement mobile.

L'objectif principal de ce travail est de démontrer la faisabilité technique d'un tel système embarqué, tout en assurant une bonne précision de détection et une réactivité suffisante pour une utilisation en conditions réelles.

Ce mémoire est structuré en trois chapitres principaux :

Le Chapitre 1 présente les fondements du projet, avec une définition de la somnolence, les méthodes de détection existantes, et l'intérêt de l'approche par vision par ordinateur. Il introduit également les techniques de traitement d'image utilisées pour analyser les yeux.

Le Chapitre 2 est consacré à la mise en œuvre de la détection à l'aide du modèle YOLOv11. Il détaille l'architecture YOLO, l'évolution des versions jusqu'à YOLOv11, la création de la base de données, son annotation avec Roboflow, l'entraînement du modèle, ainsi que l'exportation du fichier final.

Le Chapitre 3 présente l'intégration et le déploiement du modèle. Il couvre les tests locaux sous Python avec interface Tkinter, la conversion du modèle vers TensorFlow Lite, le développement de l'application mobile en Kotlin, et enfin les résultats expérimentaux, les limitations rencontrées, et les perspectives d'évolution.

Chapitre 1 : Somnolence et vision par ordinateur

1.1. Introduction à la somnolence

La somnolence est un état intermédiaire entre l'éveil et le sommeil, que l'on ressent souvent après une nuit trop courte ou une journée particulièrement éprouvante. Elle se manifeste par une baisse progressive de l'attention, un ralentissement des réflexes. Ce phénomène peut survenir discrètement, rendant sa détection difficile sans outils adaptés.

D'un point de vue biologique, la somnolence entraîne des modifications dans l'activité cérébrale, un relâchement des muscles et parfois de micro-endormissements de quelques secondes. Pourtant, ces brèves pertes de conscience peuvent avoir de lourdes conséquences, notamment dans des contextes où l'attention doit rester constante, comme au volant d'un véhicule.

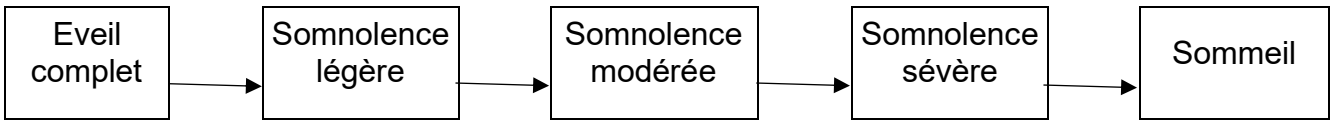

Figure 1.1: Spectre de vigilance montrant la transition entre l'éveil et le sommeil

1.2. Dangers de la somnolence dans les environnements sensibles

Dans les systèmes socio-techniques critiques, où la fiabilité humaine constitue un maillon essentiel de la chaîne de sécurité, la somnolence représente un facteur de risque majeur nécessitant une approche systémique de prévention et de détection **[1]**. Les conséquences d'une défaillance liée à la fatigue peuvent s'étendre bien au-delà de l'individu concerné, affectant la sécurité collective et engendrant des coûts socio-économiques considérables.

Tableau 1.1: Statistiques des accidents liés à la somnolence

Accidents routiers mortels (%)	Incidents aériens (%)	Erreurs médicales (%)	Risque après 24h (%)
15-20	4-7	25-35	36

Tableau 1.2: Comparaison des approches de détection de somnolence

Méthode	Précision	Temps réel	Intrusion	Coût	Applications
EEG/EOG	Très élevée	Non	Élevée	Élevé	Laboratoire, médical
ECG/HRV	Élevée	Oui	Moyenne	Moyen	Dispositifs portables
Vision par ordinateur	Élevée	Oui	Faible	Faible	Automobile, mobile
Analyse comportementale	Moyenne	Oui	Très faible	Très faible	Surveillance générale
Tests de performance	Moyenne	Non	Faible	Faible	Évaluation clinique

1.2.1. Transport routier

Le secteur des transports routiers illustre de manière exemplaire les enjeux liés à la somnolence au volant. Les données épidémiologiques révèlent que la fatigue constitue un facteur contributif dans 15 à 20% des accidents routiers mortels, avec un pic d'occurrence entre 2h et 6h du matin, ainsi qu'en début d'après-midi (14h-16h), périodes correspondant aux creux circadiens naturels **[2]**.

L'analyse comparative des performances cognitives démontre qu'un conducteur en état de somnolence présente des altérations comportementales similaires à celles observées sous l'influence de l'alcool. Ainsi, une privation de sommeil de 17 à 19 heures génère des déficits de performance équivalents à un taux d'alcoolémie de 0,05%, tandis qu'une privation de 20 à 25 heures correspond à un taux de 0,10% **[3]**.

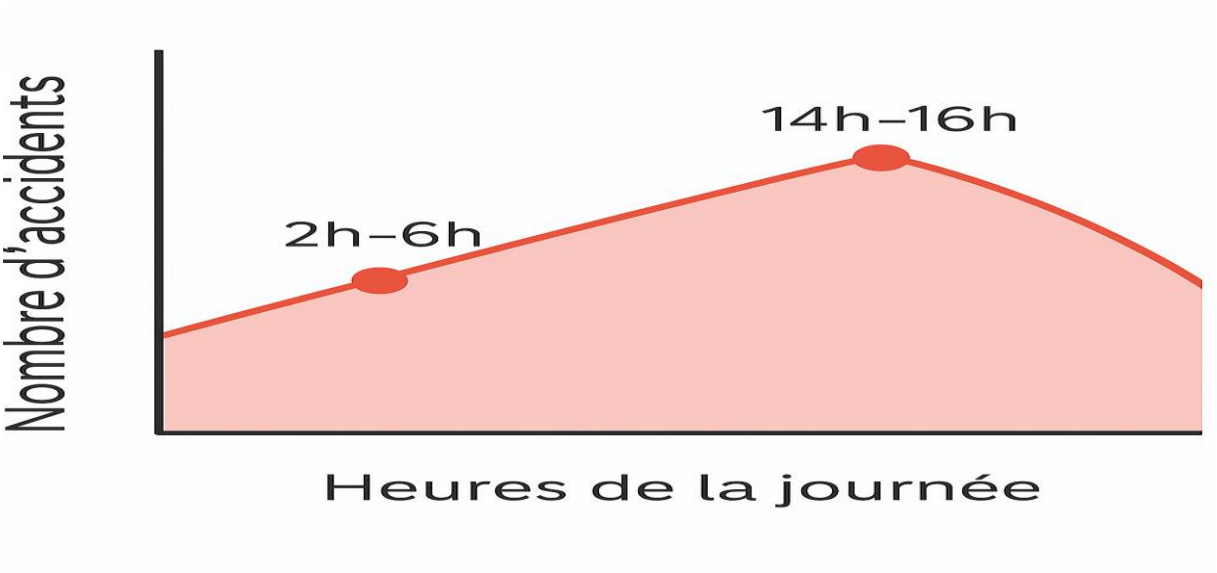

Figure 1.2: Distribution circadienne des accidents liés à la somnolence

1.2.2. Transport aérien

L'aviation civile et militaire présente des défis particuliers en matière de gestion de la fatigue, en raison de facteurs spécifiques tels que les vols long-courriers, les rotations irrégulières, les décalages horaires répétés et la pression opérationnelle inhérente à ce secteur d'activité **[4]**. L'analyse des incidents et accidents aériens révèle que la fatigue constitue un facteur causal ou contributif dans approximativement 4 à 7% des événements répertoriés par les autorités de l'aviation civile.

1.2.3. Environnement hospitalier

Le secteur hospitalier présente une problématique complexe de gestion de la fatigue, particulièrement critique en raison de l'impact direct sur la sécurité des patients. Les études épidémiologiques démontrent une corrélation significative entre la durée des gardes médicales et l'incidence des erreurs médicales, avec une augmentation de 36% du risque d'erreur grave après 24 heures de service continu **[5]**.

Figure 1.3: Corrélation entre durée des gardes et erreurs médicales

1.2.4. Industrie et infrastructures critiques

Dans le secteur industriel, particulièrement au sein des industries de process continu (pétrochimie, nucléaire, sidérurgie), la fatigue des opérateurs constitue un facteur de risque technologique majeur. L'analyse post-accidentelle de catastrophes industrielles majeures a mis en évidence le rôle contributif de la fatigue humaine dans la chaîne causale des événements **[6]**.

Tableau 1.3: Statistiques d'accidents liés à la somnolence par secteur

Secteur	Accidents (%)	Coût annuel (milliards €)	Pic Horaire	Facteurs aggravants
Transport routier	15-20	15-20	2h-6h, 14h-16h	Nuit, Monotonie
Transport aérien	4-7	5-8	Variables	Décalage horaire
Milieu hospitalier	25-35	2-3	3h-7h	Gardes prolongées
Industrie critique	10-15	8-12	2h-6h	Travail posté

1.2.5. Intérêt de la détection automatisée

C'est là que les technologies intelligentes entrent en jeu. Grâce à la vision par ordinateur et à l'intelligence artificielle, il est aujourd'hui possible de surveiller en temps réel l'état de vigilance d'une personne, sans contact ni intrusion. Ces systèmes

peuvent détecter les signes précurseurs de somnolence et alerter à temps, contribuant ainsi à prévenir les accidents.

1.3. Différentes approches de détection de la somnolence

La recherche scientifique en matière de détection de la somnolence a exploré plusieurs méthodes de mesure, chacune présentant des avantages spécifiques et des limitations. Cette diversité de méthodes reflète la complexité du phénomène de somnolence et la nécessité d'adapter les méthodes de détection aux contraintes opérationnelles spécifiques.

Tableau 1.4: Panorama des méthodes de détection de la somnolence

Approche de Détection	Physiologiques	Comportementales	Performance
Type d'approche	EEG, ECG, EOG	Vision, Clignements	Temps de réaction
Avantage	Très précis	Non-intrusif	Objectif
Inconvénient	Intrusif	Conditions variables	Détection tardive

1.3.1. Méthodes physiologiques

Les approches physiologiques constituent l'étalon-or de la mesure objective de la somnolence, s'appuyant sur l'enregistrement direct des signaux bioélectriques reflétant l'état neurophysiologique de l'individu. L'électroencéphalographie (EEG), l'électrocardiographie (ECG), l'électro-oculographie (EOG) demeurent les méthodes de référence, permettant l'analyse des rythmes cérébraux caractéristiques des différents stades de vigilance **[7]**.

Malgré leur fiabilité métrologique, ces méthodes présentent des limitations significatives pour les applications en conditions réelles : nécessité d'un équipement spécialisé, contraintes de mise en place des électrodes, inconfort pour l'utilisateur, et sensibilité aux artefacts de mouvement. Ces facteurs limitent leur utilisation aux environnements de laboratoire ou aux applications médicales spécialisées.

1.3.2. Méthodes comportementales

Ces méthodes observent des signes visibles de fatigue : clignements des yeux plus fréquents ou plus lents, bâillements, mouvements de tête, etc. Elles sont plus discrètes et adaptées aux environnements réels, mais demandent des systèmes de vision performants pour fonctionner correctement en toutes circonstances.

1.3.3. Méthodes basées sur les performances

Ici, on mesure la manière dont une personne accomplit une tâche : son temps de réaction, sa précision, sa concentration. Bien que ces méthodes offrent une évaluation objective des conséquences fonctionnelles de la somnolence, elles présentent certaines limitations : délai de détection relativement long, influence de facteurs confondants (motivation, apprentissage, stress), et nécessité d'une tâche dédiée à l'évaluation.

1.3.4. Choix de la vision par ordinateur

Dans ce projet, nous avons retenu une approche basée sur le comportement oculaire, analysé via la vision par ordinateur. Grâce aux progrès de l'apprentissage profond, ce domaine permet une détection rapide, précise, et surtout non intrusive, ce qui la rend idéale pour des applications embarquées comme les smartphones ou les véhicules.

1.4 Approche par vision par ordinateur

La vision par ordinateur est une branche de l'intelligence artificielle qui permet aux machines d'analyser, d'interpréter et de comprendre des informations visuelles extraites d'images ou de vidéos. Appliquée à la détection de la somnolence, elle permet de surveiller les comportements oculaires d'un individu afin d'identifier des signes précurseurs tels que :

- La fermeture prolongée des paupières
- Le ralentissement du clignement des yeux
- L'absence de mouvement oculaire sur une certaine durée

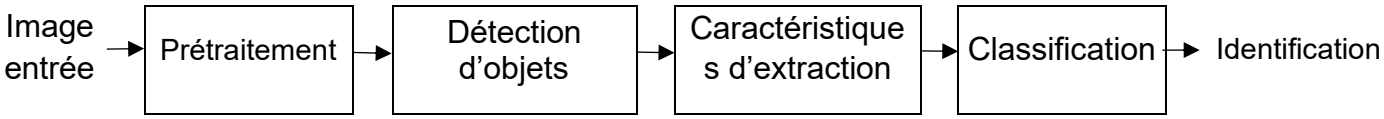

Figure 1.4: Pipeline de traitement en vision par ordinateur

L'élément central de cette approche repose sur l'utilisation de réseaux neuronaux convolutifs (Convolutional Neural Networks, ou CNN). Ces réseaux sont spécialement conçus pour le traitement d'images et se sont imposés comme l'une des techniques les plus efficaces en vision par ordinateur. Leur architecture repose sur des couches de neurones organisées de manière à extraire progressivement des caractéristiques visuelles, allant des plus simples (bords, textures) aux plus complexes (formes, objets).

Dans le cadre de la détection de somnolence, les CNN permettent de localiser précisément les yeux et d'analyser leur état en temps réel, même dans des conditions difficiles, telles que des variations de luminosité, des angles de vue inhabituels ou des visages partiellement occultés. Grâce à leur capacité à apprendre automatiquement à partir de grandes quantités de données annotées, les CNN sont capables de généraliser à de nouveaux visages ou différentes situations, ce qui les rend particulièrement adaptés aux applications embarquées.

Cette avancée rend possible le développement de systèmes de surveillance visuelle à la fois légers, rapides et non intrusifs, intégrables dans des dispositifs du quotidien comme les smartphones, les caméras embarquées dans les véhicules ou les dispositifs portables. Ainsi, la vision par ordinateur ouvre la voie à des solutions intelligentes pour la prévention des risques liés à la somnolence, notamment dans le domaine des transports.

1.5. Travaux similaires et état de l'art

La détection de la somnolence par vision est un sujet de recherche actif depuis plus d'une décennie, en raison de ses applications cruciales dans les transports, la sécurité au travail et la santé. Plusieurs approches et systèmes ont été proposés dans la littérature, que nous résumons ici pour positionner notre travail dans le cadre de l'état de l'art.

1.5.1. Paramètre PERCLOS

Introduit par Wierwille et al. **[8]**, le PERCLOS (Percentage of Eye Closure over the pupil over time) est l'un des indicateurs les plus fiables pour détecter la somnolence. Il mesure le pourcentage de temps pendant lequel les paupières couvrent plus de 80 % de la pupille pendant un intervalle donné. Des études récentes confirment que PERCLOS est l'un des indices les plus validés pour la détection passive de la somnolence, augmentant avec la privation de sommeil, après une restriction partielle du sommeil, la nuit et par d'autres manipulations de somnolence. Des études menées par la NASA ont démontré que ce paramètre était fortement corrélé avec la vigilance **[9]**.

Figure 1.5: Illustration du paramètre PERCLOS (80% de fermeture de la pupille)

1.5.2. Détection d'objets à l'aide des cascades de Haar

La méthode des cascades de Haar, introduite en 2001 par Paul Viola et Michael Jones dans leur article intitulé "Rapid Object Detection using a Boosted Cascade of Simple Features" **[10]**, a rapidement gagné en notoriété. Elle s'est distinguée par son efficacité, notamment en obtenant un taux de détection élevé sur le jeu de données complexe "MIT+CMU Dataset", tout en étant environ 15 fois plus rapide que les autres méthodes de détection faciale de l'époque.

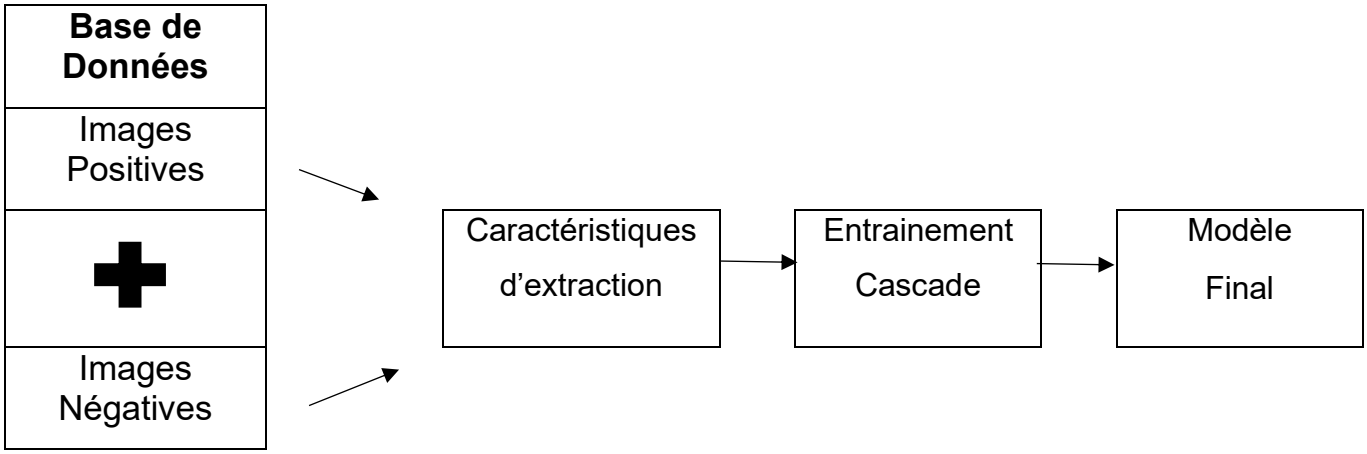

Figure 1.6: Pipeline d'entraînement des cascades de Haar

L'algorithme se décompose en quatre étapes principales :

- La création d'une base de données contenant à la fois des images positives (avec l'objet cible) et négatives (sans l'objet)
- L'extraction de caractéristiques à partir de l'ensemble des images de la base
- L'entraînement d'un classifieur en cascade basé sur ces caractéristiques
- L'application du modèle entraîné pour détecter l'objet dans de nouvelles images

1.5.3. Apport des réseaux de neurones convolutionnels (CNN)

Les CNN (Convolutional Neural Networks) représentent une sous-catégorie des réseaux de neurones, reconnue aujourd'hui comme l'une des plus performantes pour les tâches de classification d'images.

Leur fonctionnement repose sur un principe relativement simple : l'utilisateur fournit en entrée une image représentée sous forme de matrice de pixels. Cette matrice possède trois dimensions :

- Deux dimensions correspondent à la hauteur et à la largeur de l'image (notamment pour les images en niveaux de gris).
- Une troisième dimension, de profondeur 3, est utilisée pour représenter les composantes de couleur fondamentales : Rouge, Vert et Bleu (RVB).

Contrairement à un réseau de type MLP (Multi-Layer Perceptron), qui ne comporte qu'une phase de classification, l'architecture d'un CNN se divise en deux grandes parties : une partie convolutive et une partie de classification.

1. **Partie convolutive**
   Cette première étape vise à extraire les caractéristiques pertinentes de l'image tout en réduisant sa taille. Pour ce faire, l'image traverse une série de filtres qui génèrent des images intermédiaires appelées *cartes de convolution*. Ces cartes mettent en évidence des motifs spécifiques (bords, textures, etc.). Elles sont ensuite regroupées sous forme d'un vecteur de caractéristiques, souvent appelé *code CNN*.

2. **Partie classification**
   Le code CNN généré est ensuite transmis à une seconde partie composée de couches entièrement connectées, formant un perceptron multicouche (MLP). Cette étape combine les différentes caractéristiques extraites afin de prédire la classe à laquelle appartient l'image.

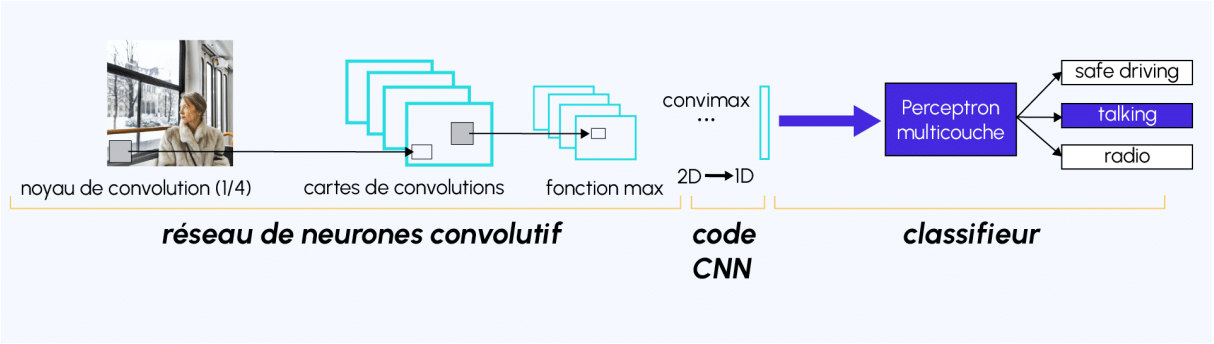

Figure 1.7: Schéma représentant l'architecture d'un CNN

Avec l'émergence des CNN (Convolutional Neural Networks), la performance en classification d'images s'est considérablement améliorée. Des architectures comme LeNet **[11]**, AlexNet **[12]**, VGG **[13]**, puis ResNet **[14]** ont été adaptées pour reconnaître l'état des yeux à partir d'images extraites. La recherche récente présente une investigation complète des méthodes de détection de somnolence, avec un focus spécifique sur l'utilisation des réseaux de neurones convolutionnels (CNN) et l'apprentissage par transfert.

Tableau 1.5: Évolution des architectures CNN

Architecture	LeNet-5	AlexNet	VGG-16	ResNet-50	YOLOv11
Année	1998	2012	2014	2016	2024
Précision (%)	85	92	94	95	96

Certains travaux ont utilisé des réseaux simples pour classifier une image de l'œil (ouvert/fermé), tandis que d'autres ont proposé des modèles plus complexes capables d'analyser des séquences temporelles pour détecter des schémas de clignement.

Cependant, ces approches nécessitaient souvent une première étape manuelle ou semi-automatique d'extraction des yeux, ce qui limitait leur intégration en temps réel.

Tableau 1.6: Performance des architectures CNN pour la détection oculaire

Architecture	Année	Paramètres	FPS	Précision	Taille modèle
LeNet-5	1998	60K	1000+	85%	0.5 MB
AlexNet	2012	60M	100	92%	240 MB
VGG-16	2014	138M	50	94%	528 MB
ResNet-50	2016	25M	75	95%	102 MB
YOLOv11	2024	20M	120+	96%	85 MB

1.5.4. YOLO et détection temps réel

L'introduction de l'architecture YOLO (You Only Look Once) par Redmon et al en 2016 **[15]** a marqué un tournant dans la détection d'objets en temps réel. Contrairement aux approches comme R-CNN, qui séparent la localisation de la classification, YOLO traite l'image comme une grille et prédit directement les boîtes englobantes et les classes associées, en un seul passage.

Les versions successives (YOLOv3, v4, v5, v8, etc.) ont apporté des améliorations en précision, vitesse et efficacité énergétique **[16]**. YOLOv11 introduit de nouvelles améliorations architecturales comme le bloc C3k2, SPPF, et C2PSA, rendant le modèle plus efficace dans l'extraction et le traitement des caractéristiques et améliorant l'attention sur les zones clés d'une image.

La version YOLOv11 que nous avons utilisée dans ce travail bénéficie des dernières optimisations en matière de backbone (utilisation de CSPDarknet, de SPPF, de modules PANet), ce qui lui confère une très bonne capacité de généralisation, même avec des jeux de données réduits. Le YOLOv11 permet ainsi de bien comprendre le CSP Darknet, SPPF et PANet.

Elle permet ainsi :

- Une détection précise des yeux malgré les variations d'angle, de distance ou de lumière
- Une classification directe de leur état (ouvert/fermé)
- Un traitement temps réel compatible avec les plateformes embarquées (Android, Raspberry Pi, etc.)
- Une optimisation facile pour l'export vers des formats légers comme TensorFlow Lite, utilisé dans notre projet pour le déploiement mobile

1.5.5. Positionnement du projet

Notre système se distingue par l'intégration complète du cycle : création d'un jeu de données annoté (via Roboflow), entraînement personnalisé du modèle YOLOv11, conversion vers un modèle optimisé (.tflite), et déploiement dans une application Android développée en Kotlin. Contrairement à de nombreux travaux qui se concentrent uniquement sur la phase d'apprentissage, nous avons mis l'accent sur la portabilité et l'utilisation concrète du système sur un terminal mobile.

1.6. Traitement d'images appliqué à la détection de la somnolence

Le traitement d'images constitue l'un des maillons fondamentaux de la chaîne de détection. Il permet de transformer les données brutes en informations exploitables pour les modèles d'apprentissage. Les étapes principales comprennent :

1.6.1. Prétraitement des images

Les images collectées doivent être redimensionnées, recadrées et normalisées afin d'assurer une cohérence dans les données d'entrée. Le recadrage se concentre sur les régions d'intérêt (ROI), principalement les yeux. La normalisation, quant à elle, consiste à mettre les valeurs de pixels dans un intervalle commun (par exemple, entre 0 et 1) pour faciliter l'apprentissage.

1.6.2. Détection des régions d'intérêt

L'identification précise de la position des yeux dans chaque image est essentielle. Des algorithmes de détection comme les classificateurs en cascade de Haar ou des CNN spécialisés sont utilisés pour extraire ces zones de manière robuste.

1.6.3. Annotation des données

Une base de données annotée est indispensable pour l'apprentissage supervisé. Chaque image est étiquetée manuellement ou semi-automatiquement, indiquant si les yeux sont ouverts ou fermés. Ces annotations servent ensuite de référence pour entraîner le modèle.

1.6.4. Classification des états oculaires

Après extraction des yeux, ceux-ci sont classifiés selon leur état. YOLOv11, combiné à un classificateur de type CNN, permet une détection et une classification simultanées en temps réel, ce qui est un atout pour les applications mobiles ou embarquées.

1.6.5. Intégration dans un système embarqué

La rapidité et la précision de YOLOv11 en font un candidat idéal pour l'intégration dans des dispositifs mobiles. L'optimisation via TensorFlow Lite permet une exécution sur des plateformes Android, ouvrant la voie à des applications de surveillance en temps réel sur smartphone ou dans les tableaux de bord de véhicules.

1.7. Intégration avec YOLOv11

En combinant ces étapes de traitement d'images avec des modèles de détection avancés comme YOLOv11, il est possible de créer des systèmes de détection de somnolence non seulement précis, mais aussi rapides et efficaces. YOLOv11 marque un avancement pivot dans la détection d'objets, combinant vitesse, précision et efficacité grâce à des innovations comme les blocs C3K2 pour l'extraction de caractéristiques et l'attention C2PSA pour se concentrer sur les régions critiques de l'image.

YOLOv11, grâce à sa capacité à détecter les objets en temps réel et à sa rapidité d'exécution, permet de traiter des flux vidéo en continu et de détecter instantanément l'état des yeux. Cela permet de déployer des systèmes embarqués légers et réactifs, parfaitement adaptés aux applications mobiles ou intégrées dans des systèmes de sécurité à bord de véhicules.

Ainsi, le traitement d'image joue un rôle clé dans le pipeline global de détection de la somnolence, en permettant l'extraction et l'analyse des données visuelles nécessaires pour prendre des décisions en temps réel, et ce, avec une efficacité optimale.

1.8. Conclusion

La somnolence représente un risque critique dans les environnements à haute vigilance où une baisse d'attention de quelques secondes peut avoir des conséquences catastrophiques. Face aux limites des méthodes traditionnelles (intrusion, coût élevé, manque de réactivité), notre approche s'appuie sur la vision par ordinateur pour offrir une solution non intrusive, temps réel et optimisée pour le mobile. Ce système intègre de l'annotation des données via Roboflow au déploiement embarqué via TensorFlow Lite, permettant une bonne détection des états oculaires (ouvert/fermé) et une alerte précoce des épisodes de somnolence.

Chapitre 2 : Entraînement du modèle et détection avec YOLOv11

2.1 Introduction à la détection d'objets

La détection d'objets est une technique de vision par ordinateur qui permet d'identifier et de localiser des objets dans une image ou une vidéo.

La localisation d'images est le processus d'identification de l'emplacement correct d'un ou de plusieurs objets à l'aide de boîtes de délimitation, qui correspondent à des formes rectangulaires autour des objets. Ce processus est parfois confondu avec la classification ou la reconnaissance d'images, qui vise à prédire la classe d'une image ou d'un objet dans une image dans l'une des catégories ou classes.

Imaginez que vous parcouriez une photo et que, d'un simple coup d'œil, vous pointiez du doigt chaque élément intéressant : « voilà une voiture, juste là un piéton, ici un feu tricolore ». C'est exactement ce que cherche à faire la détection d'objets : non seulement reconnaître quels objets sont présents dans une image, mais également indiquer précisément où ils se trouvent à l'aide de boîtes englobantes. Elle va donc bien plus loin que la classification d'images classique, qui se contente de dire « il y a une voiture quelque part » sans préciser sa position.

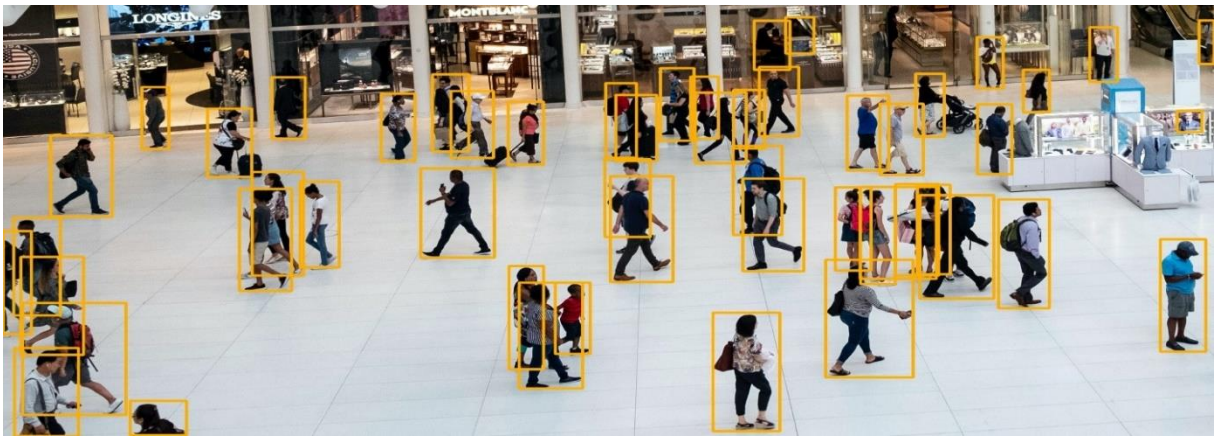

Figure 2.1: Exemple de détection d'objets dans une image urbaine

2.1.1 Méthodes traditionnelles

Avant l'arrivée de l'apprentissage profond, on misait sur des descripteurs "faits main" : Haar, HOG, SIFT… Ces signatures mathématiques tentaient de capturer la texture ou les contours, puis un classifieur (souvent un SVM) décidait si un objet était présent. Cette chaîne était fragile : chaque étape devait être optimisée séparément, et la moindre variation de luminosité ou de perspective pouvait tout dégrader.

Avec les réseaux de neurones convolutifs (CNN) **[21]**, la donne a changé. Un seul réseau apprend simultanément quoi regarder et comment le reconnaître : les premières couches découvrent des motifs simples (bords, couleurs), les suivantes composent des formes plus complexes (yeux, roues), et les dernières identifient l'objet complet tout en affinant la localisation. Des architectures comme Faster R-CNN, SSD ou YOLO **[17]** ont popularisé ce paradigme « tout-en-un », apportant une précision et une vitesse d'inférence inédites.

2.1.2 Base technologique pour des applications critiques

La détection d'objets est devenue un pilier dans des domaines aussi variés que :

Conduite autonome : percevoir la route, anticiper les obstacles, distinguer panneaux et piétons.

Surveillance intelligente : repérer des comportements suspects ou compter des personnes en temps réel.

Robotique : permettre aux robots de saisir des objets ou de naviguer dans des environnements encombrés.

Médecine : localiser des tumeurs sur des radiographies ou segmenter des organes lors d'opérations assistées.

Dans ce mémoire, nous exploitons ces avancées pour un défi très spécifique : détecter l'état des yeux d'un conducteur. Les paupières mi-closes ou fermées sont un indicateur direct de fatigue ; en repérant rapidement ces signes, notre système peut alerter avant que la somnolence n'entraîne un accident. La détection d'objets devient ainsi un outil de sécurité proactive, transformant une caméra embarquée en sentinelle numérique qui veille sur la vigilance du conducteur.

2.2 Architecture et principes de YOLO

Les frameworks de détection antérieure ont examiné différentes parties de l'image plusieurs fois à différentes échelles et ont réorienté la technique de classification des images pour détecter les objets. Cette approche est lente et inefficace.

YOLO adopte une approche totalement différente **[17]**. Il ne regarde l'image entière qu'une seule fois et passe par le réseau une fois et détecte les objets. D'où le nom. Il est très rapide. C'est la raison pour laquelle il est devenu si populaire.

2.2.1 Principe de fonctionnement de l'algorithme YOLO

L'algorithme YOLO (You Only Look Once) est une méthode de détection d'objets en temps réel qui repose sur une approche innovante et unifiée : il traite l'image d'un seul coup pour détecter à la fois la position des objets (via des cadres de délimitation) et leur catégorie **[17]**. Ce paradigme contraste avec les approches classiques comme R-CNN ou Fast R-CNN, qui procèdent en plusieurs étapes (proposition de régions, classification, ajustement).

YOLO transforme la détection d'objets en un problème de régression unique, qui consiste à prédire, à partir d'une image, un vecteur contenant toutes les informations nécessaires à la détection.

2.2.2. Division de l'image en une grille

L'image d'entrée est tout d'abord divisée en une grille régulière de dimensions S×S. Par exemple, dans YOLOv1, on choisit souvent S=7, ce qui signifie que l'image est divisée en 49 cellules **[17]**.

Chaque cellule de cette grille est responsable de la détection des objets dont le centre se situe à l'intérieur de cette cellule. Cette approche repose sur le principe que chaque objet ne sera détecté qu'une seule fois, par la cellule la plus proche de son centre.

2.2.3 Impact de la granularité spatiale : le rôle crucial de la grille S×S

L'algorithme YOLO repose sur un principe fondamental : la division de l'image d'entrée en une grille régulière de dimension S×S, où chaque cellule est responsable de la détection des objets dont le centre se situe dans sa zone. Cette approche permet de transformer la détection d'objets en un problème de régression localisé, rendant possible une prédiction conjointe des coordonnées des boîtes de délimitation et des classes associées à partir d'un seul passage dans le réseau.
Cependant, le choix de la dimension S n'est pas anodin. Il représente un compromis direct entre la précision spatiale de la détection et les performances en temps réel du modèle. Une valeur de S faible (par exemple 7 ou 13) implique un nombre réduit de cellules, ce qui permet des inférences rapides, mais limite la capacité du modèle à détecter des objets de petite taille ou plusieurs objets proches. En effet, une cellule ne peut en général détecter qu'un seul objet, ce qui pose problème en présence d'objets multiples dans une même zone de l'image.
À l'inverse, une valeur de S plus élevée (par exemple 19, 26 ou davantage) augmente la résolution de la grille, et par conséquent, la finesse de la localisation. Cela améliore considérablement la détection d'objets petits ou denses, au prix toutefois d'un coût computationnel accru et d'une latence plus importante.
Les versions récentes de YOLO, notamment à partir de YOLOv3, intègrent une stratégie dite de prédiction multi-échelle. Cette dernière consiste à générer simultanément des prédictions à plusieurs résolutions (par exemple 80×80, 40×40, 20×20), permettant ainsi au modèle de capter efficacement les objets de différentes tailles. YOLOv11 pousse encore plus loin cette logique en optimisant dynamiquement les niveaux de granularité en fonction des caractéristiques de l'image d'entrée.

Dans le cadre de ce projet, qui vise la détection en temps réel de l'état des yeux du conducteur sur un terminal mobile, cette gestion fine de la grille s'avère essentielle. La petite taille relative des yeux dans l'image, combinée aux contraintes matérielles des appareils embarqués, nécessite une architecture capable de maintenir une haute précision sans sacrifier la vitesse d'exécution. La flexibilité des grilles multi-échelles de YOLOv11 permet ainsi de concilier détection précise et performance embarquée, deux exigences clés du système de détection de somnolence développé.**[18]**

.

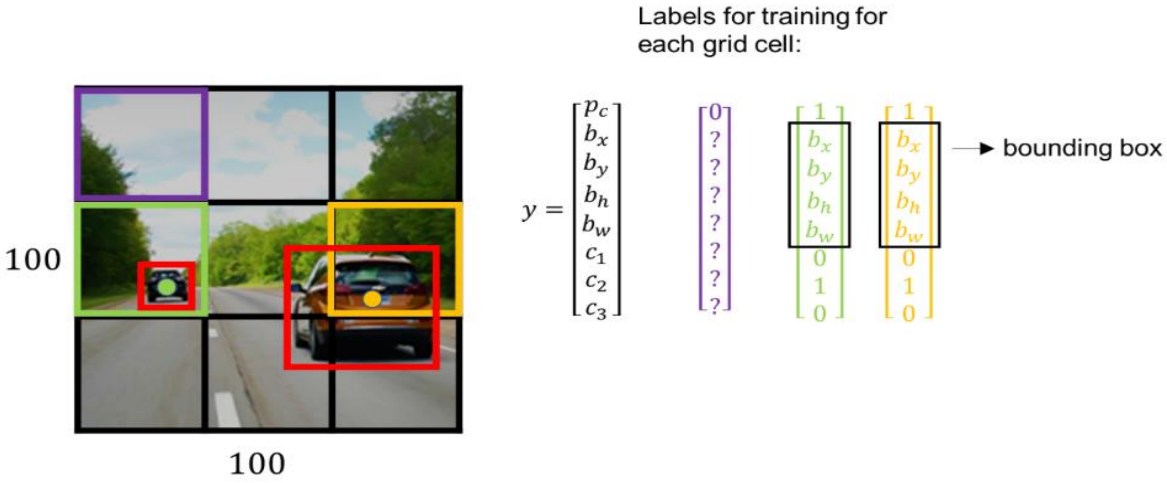

Figure 2.2: Division de l'image en grille S×S pour la détection YOLO

2.2.4. Prédictions par cellule

Chaque cellule de la grille va produire un certain nombre de prédictions. Ces prédictions concernent à la fois :

- Les boîtes englobantes (bounding boxes), qui encadrent les objets dans l'image.
- Les probabilités de classes, qui indiquent la nature de l'objet détecté.

Pour chaque cellule, le modèle prédit[17] :

Boîtes de délimitation, chacune représentée par 5 valeurs :

x,y : coordonnées du centre de la boîte, relatives à la cellule

(Valeurs entre 0 et 1),

w,h : largeur et hauteur de la boîte, relatives à la taille de l'image

(Valeurs entre 0 et 1),

Pc : score de confiance que cette boîte contient effectivement un objet et que la prédiction est correcte.

Probabilités de classe :

P(classei|objet)×P(\text{classe}_i|\text{objet})×P(classei|objet), pour chaque classe d'objet.

La sortie par cellule est donc un vecteur de dimension :

B×5+CB×5+C

Et la sortie totale pour l'image est :

S×S×(B×5+C)×S

Exemple :

Si on considère une image divisée en une grille 3×3, avec B=1 (une seule boîte par cellule) et C=3 classes (par exemple *voiture*, *camion*, *bus*), alors chaque cellule produit :

1×5+3=8 valeurs

Et la sortie globale est :

3×3×8=72

2.2.5. Sémantique des valeurs prédites

Pour chaque boîte prédit par une cellule **[17]** :

Pc : c'est un indicateur de confiance (entre 0 et 1). Il est élevé si la boîte contient effectivement un objet, et si cette boîte prédite correspond bien à la boîte réelle (mesuré par l'IoU – Intersection over Union).

x,y : position relative du centre de l'objet dans la cellule (entre 0 et 1).

w,h : taille de la boîte, exprimée comme une proportion de la taille de l'image.

P(classei|objet)×P(\text{classe}_i|\text{objet})×P(classei|objet):probabilités conditionnelles des classes, données par un softmax.

Le produit :

Pc×P(classei|objet)P_c \times P(\text{classe}_i \mid \text{objet})Pc×P(classei|objet)

Représente la probabilité finale que l'objet de classe i soit présent dans la cellule.

2.2.6. Illustration par un cas concret

Prenons un exemple simplifié avec une grille 3×33 \times 33×3 et un seul objet par cellule. Supposons que deux voitures soient présentes dans l'image. Le centre de la première voiture se situe dans la cellule verte, celui de la seconde dans la cellule jaune. La cellule violette est vide.

La cellule verte aura :

Pc≈1, P_c=1Pc≈1

Une boîte bien localisée autour de la voiture

Une probabilité élevée pour la classe *voiture*

La cellule jaune aura un comportement similaire pour l'autre voiture.

La cellule violette, en revanche, prédit Pc≈0P_c=0Pc≈0 et des probabilités de classe faibles, car aucun objet n'est présent dans cette zone.

Cette configuration est illustrée dans la figure suivante :

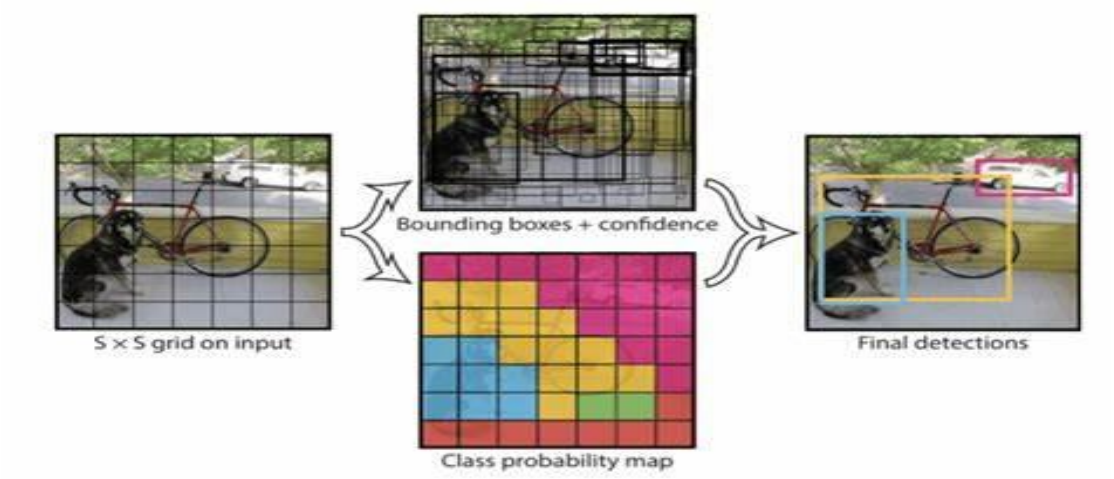

Figure 2.3: Exemple de prédiction YOLO sur grille avec détection multi-objets

2.2.7. Entraînement du modèle et rôle des étiquettes (labels)

Lors de l'entraînement, le modèle apprend à prédire ces vecteurs en comparant ses sorties aux annotations réelles fournies sous forme de bounding boxes. Ces boîtes, tracées manuellement ou via un outil comme Roboflow, représentent la vérité terrain (ground truth).

La fonction de coût (loss function) utilisée dans YOLO combine **[17]** :

Une erreur de localisation sur x,y,w,h;

Une erreur de confiance sur Pc;

Une erreur de classification (souvent une entropie croisée).

Le modèle apprend ainsi à produire des boîtes aussi proches que possible des vraies boîtes rouges (annotations), en optimisant simultanément la localisation et la classification.

2.2.8. Avantages de cette approche unifiée

Rapidité : YOLO traite toute l'image en une seule passe → détection en temps réel **[17]**.

Simplicité : une seule architecture end-to-end.

Globalité : le modèle apprend à détecter dans le contexte global de l'image, ce qui réduit les faux positifs.

2.2.9. Architecture

L'architecture du modèle se compose de 24 couches convolutives pour extraire les features suivies de 2 couches denses entièrement connectées pour réaliser la détection d'objets [**17]**.

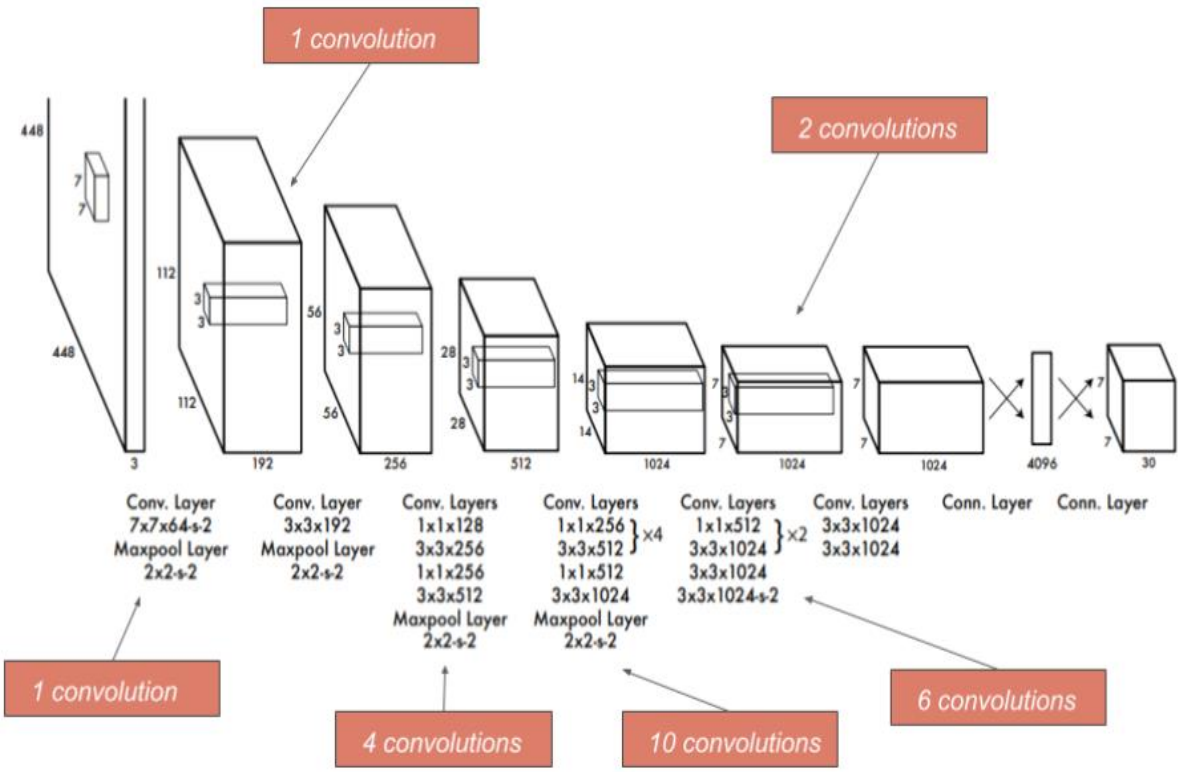

Figure 2.4: Architecture du réseau YOLOv1 avec backbone convolutionnel

1. Pré-traitement de l'image

Redimensionnement : L'image d'entrée est redimensionnée à 448x448 pixels pour standardiser la taille des inputs **[17]**.

Normalisation : Les valeurs des pixels sont généralement normalisées

(ex. : [0, 255] → [0, 1]).

2. Backbone convolutionnel (décrit dans l'image)

Convolutions 1x1 → 3x3 :

1x1 : Réduction de dimensionnalité (ex. : 512 canaux → 256 canaux) pour diminuer la complexité calculatoire.

3x3 : Extraction de features spatiales (bords, textures, motifs).

Activation : ReLU après chaque couche (sauf la dernière).

MaxPooling 2x2+2 :

Réduction de la résolution spatiale (divise par 2) tout en augmentant la profondeur (nombre de filtres).

Structure empilée :

Blocs répétés avec augmentation progressive des filtres (192 → 1024) pour capturer des features hiérarchiques.

Techniques de régularisation

Normalisation par lots (Batch Norm) :

Appliquée après chaque convolution (avant ReLU) pour stabiliser l'apprentissage.

Réduit les dépendances aux initialisations et accélère la convergence.

Dropout :

Désactive aléatoirement des neurones pendant l'entraînement pour éviter le surajustement (ex. : taux de 0.5).

3. Fonctions d'activation

ReLU (Rectified Linear Unit) :

Utilisée partout sauf en sortie : f(x) = max (0, x).

Évite le problème de gradient vanishing et accélère l'entraînement [**21**].

Linéaire (dernière couche) :

Pas de non-linéarité pour permettre des prédictions de sortie non contraintes (utile pour les coordonnées de bounding box).

4. Sortie du modèle

Format de prédiction :

Si c'est un modèle de type YOLO, la sortie est un tenseur de forme (S, S, B*(5 + C)) :

- S : Grille de sortie (ex. : 7x7 pour YOLOv1).
- B : Nombre de boîtes prédites par cellule.
- 5 : (x, y, w, h, confidence).
- C : Probabilités de classe.

Post-traitement :

- Suppression des boîtes redondantes (NMS).
- Filtrage par score de confiance.

En général YOLO divise l'image d'entrée en une grille régulière (ex. : 13×13). Chaque cellule de cette grille est responsable de prédire :

- Un certain nombre de boîtes englobantes (souvent 3 à 5);
- Un score de confiance (indiquant la probabilité qu'un objet soit présent);
- Les probabilités d'appartenance à chaque classe.

Les coordonnées des boîtes sont exprimées sous forme relative à la cellule, ce qui permet de localiser finement les objets.

L'architecture se compose généralement de trois modules principaux :

- **Backbone :** réseau convolutif chargé d'extraire les caractéristiques visuelles.
- **Neck :** structure intermédiaire permettant de fusionner les informations multi-échelles.
- **Head :** couche finale réalisant les prédictions de position, classe et score.

2.3 Évolution des versions YOLO jusqu'à YOLOv11

Depuis son introduction en 2016 avec YOLOv1, ce modèle de détection d'objets n'a cessé de se perfectionner pour relever des défis toujours plus ambitieux. Version après version, des améliorations notables ont été apportées, boostant ses performances, sa fiabilité et sa polyvalence.

Récapitulatif de l'évolution

A. YOLOv1

YOLOv1 (You Only Look Once, 2016) a marqué une rupture dans le domaine de la détection d'objets en temps réel grâce à son approche innovante **[17]**. Contrairement aux méthodes traditionnelles (comme les réseaux à régions proposées par R-CNN), YOLOv1 traite la détection comme un problème de régression unique, permettant de prédirect les boîtes englobantes et les classes en une seule passe du réseau.

Fonctionnement de YOLOv1

Division de l'image : L'image est découpée en une grille (ex: 7×7). Chaque cellule est responsable de prédire les objets dont le centre se trouve dans sa zone **[17]**.

Prédiction des boîtes : Pour chaque cellule, le modèle prédit :

B boîtes englobantes (coordonnées *x, y, w, h* + score de confiance).

Classes probabilistes (via un softmax sur les catégories).

Fusion des prédictions : Les boîtes redondantes sont éliminées via un Non-Maximum Suppression (NMS).

- **Avantages**

-Rapidité : Traitement en temps réel (> 45 FPS) grâce à une architecture unifiée (basée sur CNN) **[17]**.

-Vision globale : Contrairement aux méthodes par fenêtres glissantes, YOLO analyse toute l'image simultanément, réduisant les faux positifs.

- **Limites**

-Faible précision sur petits objets (dû à la grille grossière).

-Difficulté avec les objets proches ou groupés (1 seule classe par cellule).

- **Architecture**

Un CNN inspiré de *GoogLeNet* (24 couches de convolution + 2 couches fully connected), avec des couches de normalisation pour stabiliser l'apprentissage [**17]**.

B. YOLO V2 et V3

I. YOLOv2 (2017) - "YOLO9000: Better, Faster, Stronger"

- **Architecture**

-Backbone: Darknet-19 (19 couches) **[22]**;

-Utilisation de Batch Normalization sur toutes les couches;

-Classificateur haute résolution (448×448 px).

- **Innovations techniques**

-Introduction d'Anchor Boxes (5 ancres optimisées par k-means) **[22]**;

-Entraînement multi-échelle (résolution variable);

-Mécanisme de pas direct (passthrough layer) pour détection fine.

- **Performances**

-78.6 mAP sur PASCAL VOC 2007**[22]**;

-40 FPS à 544×544 px;

-Détection de plus de 9000 classes (YOLO9000).

II. YOLOv3 (2018)

- **Architecture améliorée**

-Backbone: Darknet-53 (53 couches avec connexions résiduelles) **[19]**;

-Détection à 3 échelles (8×8, 16×16, 32×32).

Améliorations techniques

-9 Anchor Boxes (3 par échelle) **[19]**;

-Remplacement du softmax par une classification binaire;

-Meilleure gestion des petits objets.

- **Performances**

-60.6 mAP sur COCO **[19]**;

-58 FPS à 608×608 px;

-Robustesse accrue aux situations complexes.

- **Comparaison technique**

Tableau 2.1: Comparaison entre les caractéristiques de YOLOv2 & YOLOv3

Caractéristique	YOLOv2	YOLOv3
Architecture	Darknet-19	Darknet-53
Boîtes d'ancrage	5	9 (3 par échelle)
Détection multi-échelle	Optionnelle	Intégrée (3 niveaux)
Précision (mAP COCO)	~55	60.6
Vitesse (FPS)	40	58

Ce tableau compare YOLOv2 et YOLOv3 selon plusieurs points importants.

- Architecture:
  YOLOv2 utilise une architecture plus simple (Darknet-19), tandis que YOLOv3 utilise une architecture plus avancée (Darknet-53), ce qui permet de mieux détecter les objets.

- Boîtes d'ancrage:
  YOLOv2 utilise 5 boîtes pour détecter les objets. YOLOv3 en utilise 9 réparties sur 3 niveaux, ce qui améliore la précision, surtout pour les petits objets.

- Détection multi-échelle:
  YOLOv2 ne fait pas toujours de détection à plusieurs échelles. YOLOv3, lui, le fait automatiquement, ce qui permet de mieux détecter des objets de tailles différentes.

- Précision(mAP):
  YOLOv3 est plus précis que YOLOv2 : environ 60.6% contre 55%.
- Vitesse(FPS):
  YOLOv3 est aussi plus rapide (58 images par seconde) que YOLOv2 (40 FPS), malgré une structure plus complexe.

- **Évolution majeure**

-YOLOv2 a introduit les concepts fondamentaux (ancres, normalisation) **[22]**;

-YOLOv3 a perfectionné l'approche (architecture résiduelle, détection multi-échelle) **[19]**;

-Les deux versions ont établi l'équilibre vitesse/précision caractéristique de YOLO.

C. YOLOv4

- **Contexte et Contributions Majeures**

Développé par Alexey Bochkovskiy (sans Joseph Redmon), YOLOv4 se positionne comme une optimisation systématique des techniques existantes pour **[23]** :

-Maintenir la vitesse tout en boostant la précision;

-Rendre l'entraînement plus accessible (avec un seul GPU);

-Standardiser les "tricks" de training efficaces.

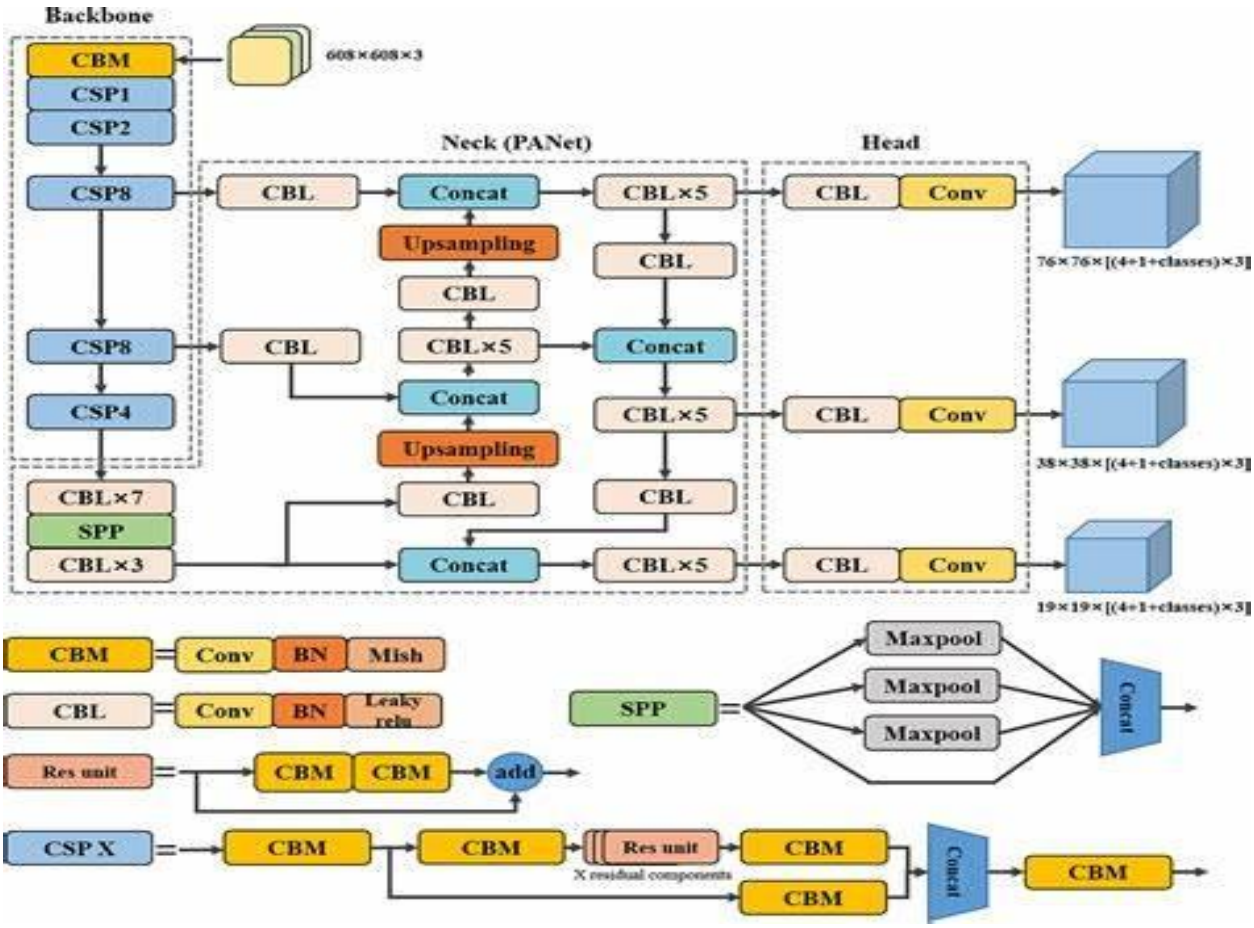

Figure 2.5: Architecture YOLOV4

Cette figure montre l'architecture d'un réseau de neurones pour la détection d'objets, organisé en 3 parties principales :

- **. Backbone (gauche)** : Extrait les caractéristiques de l'image d'entrée avec des couches CSP de plus en plus profondes.
- **Neck/PANet (centre)** : Combine les caractéristiques de différentes tailles en utilisant :

i. Upsampling pour agrandir les petites cartes

ii. Concatenation pour fusionner les informations

iii. Connexions entre tous les niveaux

- **Head (droite)** : Produit 3 sorties de détection à différentes échelles :

i. 76×76 (objets petits)

ii. 38×38 (objets moyens)

iii. 19×19 (objets grands)

Blocs de base (bas) :

i. CBM/CBL : Convolution + normalisation + activation
ii. SPP : Pooling à plusieurs échelles
iii. CSP : Blocs avec connexions résiduelles

- **Innovations Techniques Clés**

i. Architecture (Backbone + Neck + Head)

Backbone : CSPDarknet53**[23]**
- Variante améliorée de Darknet53 avec *Cross Stage Partial connections* pour réduire les calculs redondants.

Neck : PANet + SPP (Spatial Pyramid Pooling)
Meilleure agrégation des features multi-échelles.

-Head : Même principe que YOLOv3 mais optimisé.

ii. Optimisations Clés

Data Augmentation :

-Mosaic augmentation (combinaison de 4 images) **[23]**;

-Self-Adversarial Training (SAT).

3. Avantages Pratiques

Entraînement possible avec un seul GPU (contre 4+ pour les versions précédentes) **[23]**;

Compatibilité avec les frameworks populaires (TensorFlow, PyTorch via Darknet);

Meilleure détection des petits objets grâce au PANet.

4. Limitations

-Complexité accrue (hyperparamètres sensibles);

-Besoin en ressources toujours élevé pour le fine-tuning.

5. Impact Industriel

YOLOv4 a été largement adopté pour :

Les systèmes de vidéosurveillance temps réel;

L'analyse drone/satellite;

Les applications embarquées (avec des versions light comme YOLOv4-tiny).

YOLOv4 marque un tournant vers l'optimisation *pratique* des détecteurs, combinant recherche académique et besoins terrain **[23]**.

D. YOLOv5

i. Origine et Caractéristiques

Développé par Ultralytics**[20][24]**;

Première version YOLO basée sur PyTorch;

Focus sur l'efficacité industrielle plutôt que l'innovation algorithmique.

ii. Innovations Techniques

- **Architecture Principale**

-Backbone CSP amélioré;

-Mécanisme PANet optimisé;

-Système d'ancres auto-apprenant.

- **Fonctionnalités Clés**

-Calcul automatique des ancres (k-means intégré);

-Pré-traitement d'image adaptatif;

-Augmentation de données embarquée;

-Export multi-plateforme natif.

- **Avantages Pratiques**

-Entraînement 2-4x plus rapide que les versions précédentes **[24]**;

-Modèle léger (à partir de 14MB pour YOLOv5s);

-Configuration automatique des hyperparamètres;

-Documentation complète pour le déploiement.

- **Débats et Limitations**

-Nom controversé (perçu comme marketing);

-Absence de publication académique officielle;

-Performances légèrement inférieures aux dernières versions YOLO.

- **Adoption Industrielle**

-Large utilisation dans les applications edge;

-Intégration avec les principales plateformes cloud;

-Solution privilégiée pour les projets temps réel.

YOLOv5 marque une transition vers des implémentations plus accessibles, favorisant l'adoption industrielle au détriment des avancées algorithmiques majeures. Son succès tient principalement à sa facilité d'utilisation et son intégration avec les écosystèmes modernes de machine learning **[24]**.

E. YOLOv6 à YOLOv8

- **Contexte et Enjeux**

La période 2022-2023 a marqué une diversification significative des architectures YOLO, avec l'émergence de branches parallèles répondant à des besoins spécifiques :

-YOLOv6 (Meituan) : Optimisation pour l'industrie **[25]**.

-YOLOv7 (Academia) : Innovations algorithmiques **[26]**.

-YOLOv8 (Ultralytics) : Approche polyvalente **[27]**.

- **Analyse Comparative**

  i. **Innovations Architecturales**

  YOLOv6 se distingue par son EfficientRep, un backbone spécialement conçu pour les plateformes embarquées, combiné à un mécanisme Rep-PAN optimisant l'utilisation des ressources matérielles **[25]**. La perte SIoU améliore significativement la précision de localisation.

YOLOv7 introduit le concept révolutionnaire de E-ELAN, permettant un scaling dynamique du modèle **[26]**. Son "Bag of Freebies" regroupe des techniques d'augmentation de données à coût computationnel nul, tandis que le "Bag of Specials" optimise les modules critiques.

YOLOv8 marque une rupture avec l'adoption d'une architecture anchor-free, simplifiant considérablement le pipeline de détection **[27]**. Son CSPDarknet-PAFPN et sa tête task-specific en font un couteau-suisse pour les applications industrielles complexes.

ii. Performances réelles

Les benchmarks COCO révèlent une progression constante :

-Gain de +3.4 points de mAP@0.5 entre v5 et v8;

-Amélioration de +10 FPS sur la même hardware (V100);

-Réduction de 45% de la taille des modèles (v6 vs v5);

iii. Cas d'Usage Spécifiques

-Secteur Automobile : YOLOv6 excelle dans les systèmes ADAS **[25]**;

-Recherche Médicale : YOLOv7 permet des détections précises sur images 3D **[26]**;

-Robotique Industrielle : YOLOv8 domine pour les applications multi-tâches **[27]**.

- **Perspectives Futures**

L'évolution récente montre trois tendances lourdes :

Démocratisation : Simplification des pipelines pour les non-experts.

Spécialisation : Adaptation aux niches applicatives.

Convergence : Unification des tâches de vision.

Les prochaines versions devraient intégrer des mécanismes d'attention et potentiellement des approches hybrides CNN-Transformers, tout en maintenant l'efficacité caractéristique des YOLO.

- **Implications Pratiques**

Pour les praticiens :

-Startups : Privilégier YOLOv8 pour sa polyvalence **[27]**.

-Industrie Lourde : Opter pour YOLOv6 en environnement contraint **[25]**.

-Chercheurs : Explorer YOLOv7 pour ses innovations algorithmiques **[26]**.

Cette évolution reflète la maturité croissante du domaine, où l'accent se déplace des simples gains de performance vers l'adoption à grande échelle et l'intégration dans des systèmes complexes.

F. YOLOv11:

Vue d'ensemble

YOLO11 est la dernière itération de la série des détecteurs d'objets en temps réel. Ultralytics YOLO de détecteurs d'objets en temps réel, redéfinissant ce qui est possible avec une précision, une vitesse et une efficacité de pointe **[28]**. S'appuyant sur les avancées impressionnantes des versions précédentes de YOLO, YOLO11 apporte des améliorations significatives à l'architecture et aux méthodes d'apprentissage, ce qui en fait un choix polyvalent pour un large éventail de tâches de vision par ordinateur.

Caractéristiques principales

Extraction améliorée des caractéristiques : YOLO11 utilise une architecture améliorée de l'épine dorsale et du cou, qui renforce les capacités d'extraction des caractéristiques pour une détection plus précise des objets et l'exécution de tâches complexes **[28]**.

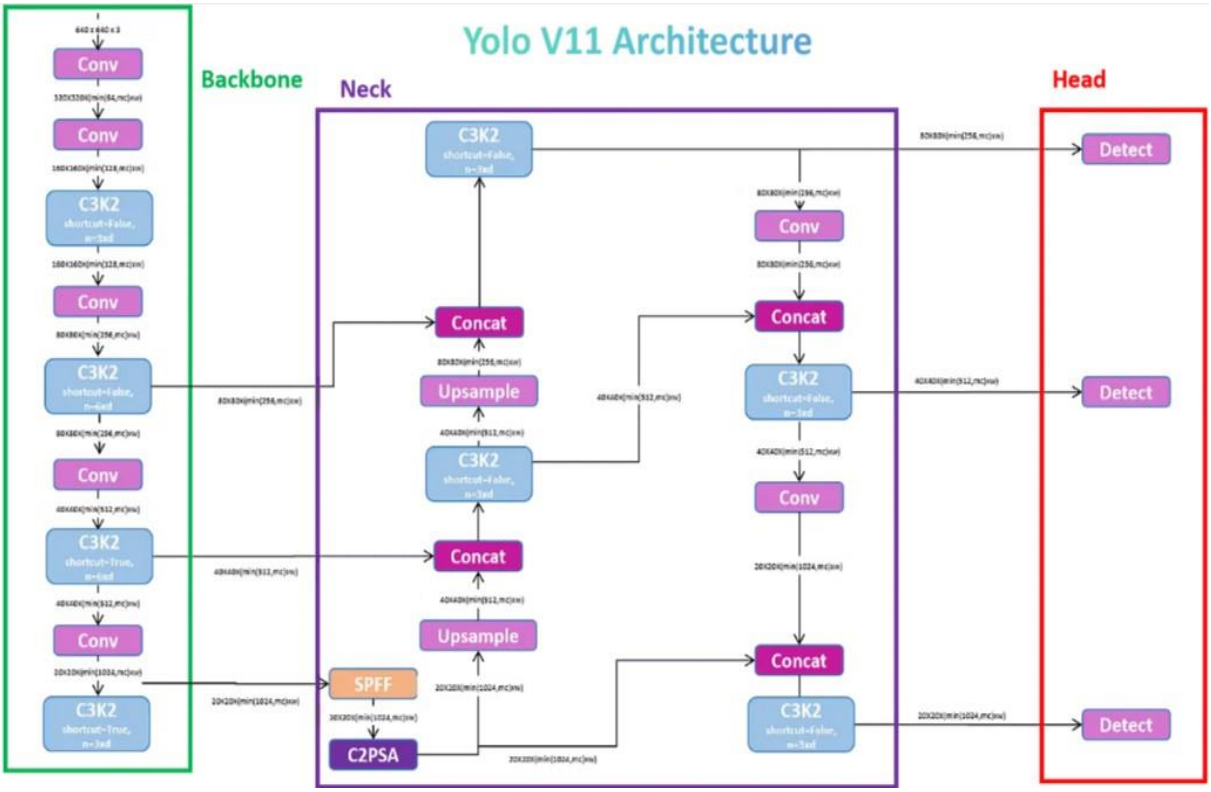

Figure 2.6 : Architecture YOLOv11

Cette figure présente l'architecture de **YOLO v11**, un détecteur d'objets moderne structuré en trois parties :

- **Backbone (vert)**

Extrait les caractéristiques hiérarchiques avec une alternance de couches :

i. **Conv** : Convolutions standard
ii. **C3K2** : Blocs avec connexions résiduelles
iii. Réduction progressive de la résolution spatiale

- **Neck (violet)**

Implémente un réseau de fusion multi-échelle avec :

i. **SPPF** : Spatial Pyramid Pooling rapide pour capturer différentes échelles
ii. **C2PSA** : Module d'attention spatiale avancé
iii. **Upsample** : Remontée en résolution
iv. **Concat** : Fusion des caractéristiques de différents niveaux
v. Architecture bidirectionnelle (top-down et bottom-up)

- **Head (rouge)**

Trois têtes de détection parallèles produisant des prédictions à différentes échelles pour détecter :

i. Objets petits (haute résolution)
ii. Objets moyens
iii. Objets grands (basse résolution)

- **Innovations clés**

i. **C3K2** : Version optimisée des blocs C3 avec noyaux 3x3
ii. **C2PSA** : Attention spatiale pour améliorer la précision
iii. **SPPF** : Version plus rapide du SPP

Cette architecture permet à YOLO v11 de détecter efficacement des objets de toutes tailles avec une bonne balance entre vitesse et précision.

Optimisé pour l'efficacité et la rapidité : YOLO11 présente des conceptions architecturales raffinées et des pipelines de formation optimisés, offrant des vitesses de traitement plus rapides et maintenant un équilibre optimal entre la précision et la performance **[28]**.

Une plus grande précision avec moins de paramètres : Grâce aux progrès réalisés dans la conception du modèle, YOLO11m atteint sur l'ensemble de données COCO tout en utilisant 22 % de paramètres en moins que YOLOv8m, ce qui le rend efficace sur le plan des calculs sans compromettre la précision **[28]**.

Adaptabilité à tous les environnements : YOLO11 peut être déployé de manière transparente dans différents environnements, y compris les appareils périphériques, et les plateformes cloud et les systèmes prenant en charge les GPU NVIDIA, ce qui garantit une flexibilité maximale **[28]**.

Large éventail de tâches prises en charge : Qu'il s'agisse de détection d'objets, de segmentation d'instances, de classification d'images, d'estimation de la pose ou de détection d'objets orientés (OBB), YOLO11 est conçu pour répondre à un ensemble varié de défis en matière de vision par ordinateur **[28]**.

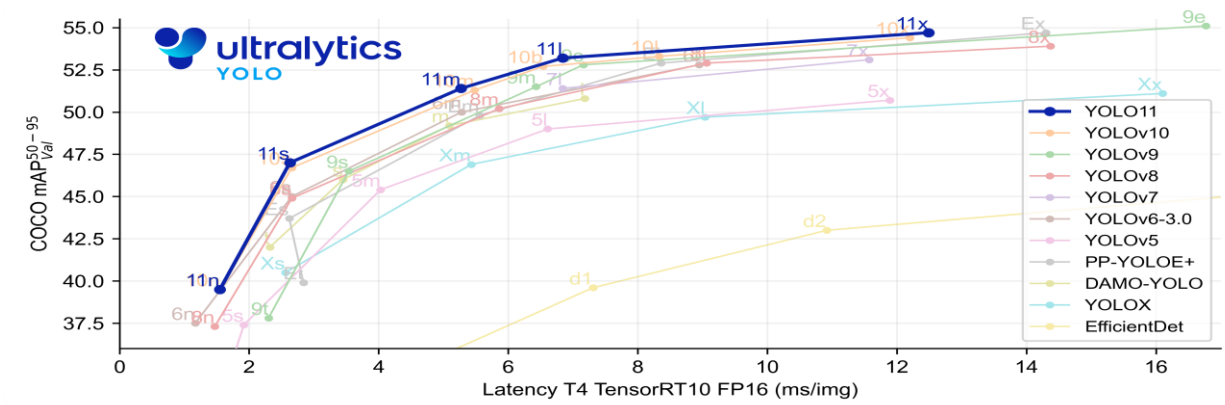

Figure 2.7: Évolution de YOLO : YOLOv11 en tête sur le benchmark COCO (mAP vs Latence)

Choix de YOLOv11 pour ce projet

L'objectif de ce projet étant de détecter l'état des yeux en temps réel sur un smartphone, le choix de YOLOv11 s'est imposé naturellement :

Il permet une détection rapide et précise des objets de petite taille.

Il est entraînable sur Google Colab avec PyTorch.

Il est facilement convertible vers TensorFlow Lite, ce qui est indispensable pour une application Android.

Il est léger et optimisé pour les déploiements sur des plateformes à faible puissance (smartphone, Raspberry Pi, etc.).

Grâce à ces caractéristiques, YOLOv11 constitue une solution technique robuste et adaptée à notre cas d'usage.

2.4 Constitution et annotation de la base de données avec Roboflow

Constitution de la base de données

Constituer une base de données pour la vision par ordinateur veut dire regrouper un ensemble d'images pertinentes pour la tâche à réaliser (détection d'objets, classification, etc.).

Étapes à faire :

a. Définir l'objectif du modèle

Avant de créer une base de données, il faut savoir :

Qu'est ce qu'il faut détecter ou reconnaître;

Les classes d'objets concernées.

b. Collection des images

Importation des photos depuis un ordinateur ou un smartphone ou bien Utiliser des images de banques libres (Unsplash, depuis un autre projet de roboflow, Open Images Dataset, Kaggle etc.) **[29]** ;

Capturer directement des images via webcam ou caméra. Dans Roboflow :

-Cliquer sur "Create New Project";

-Donner un nom au projet, et choisir le type de projet (détection, classification, etc.);

-Importation des images.

c. Organisation et nettoyage

-Suppression des images floues, mal cadrées ou hors sujet;

-Uniformation de la résolution si nécessaire;

-Classement des images en dossiers ou labels selon les classes.

d. Annotation de la base de données

L'annotation est une étape importante, car elle consiste à dire à la machine ce qu'il y a sur l'image, et où ça se trouve (dans le cas de la détection ou de la segmentation).

Types d'annotations :

-Détection d'objet → bounding boxes.

-Classification → une étiquette par image.

-Segmentation → pixel par pixel.

-Pose estimation → points clés (keypoints).

Dans Roboflow :

-Cliquer sur une image → l'éditeur d'annotation s'ouvre ;

-Dessiner un rectangle autour de l'objet à identifier ;

-Attribution d'une étiquette (ou classe) à ce rectangle ;

-Répétition pour tous les objets visibles ;

-Passer à l'image suivante.

Annotation assistée :

Le site Roboflow propose également :

-L'auto-annotation avec des modèles pré-entraînés (YOLOv8, etc.).

-L'importation de fichiers d'annotation existants (COCO, YOLO, etc.).

4. Prétraitement et Augmentation

Une fois les annotations faites :

-On peut appliquer des transformations (niveaux de gris, recadrage, flouter, etc.).

-Générer automatiquement des variations (rotation, flip, luminosité...).

-Ça permet d'augmenter la robustesse du modèle sans collecter plus d'images.

e. Export de la base de données

Quand tout est prêt :

-Cliquer sur “Generate Dataset”;

-Choisir le format de sortie (YOLOv11, COCO JSON, TensorFlow TFRecord, etc.);

Roboflow génère une base de données avec :

Un dossier train/, Un dossier valid/, Un dossier test/, et leurs fichiers d’annotation.

On obtient un lien de téléchargement ou une API de récupération directe dans les scripts d’entraînement.

2.5. Conclusion

L'architecture YOLOv11 représente l'aboutissement d'une évolution majeure des modèles de détection en temps réel, combinant vitesse, précision et efficacité énergétique. Son approche unifiée (division en grille S×S, prédictions multi-échelles, traitement end-to-end) résout les limitations des méthodes traditionnelles (fragilité aux variations, lenteur) et des versions antérieures de YOLO (granularité spatiale limitée).

Chapitre3. Implémentation & résultats

3.1 Introduction

Dans ce chapitre, nous présentons la mise en œuvre de notre système basé sur l'apprentissage profond pour la détection et l'identification de l'état des yeux. Notre étude porte sur l'évaluation des performances de YOLOv11, utilisé pour entraîner un modèle capable de détecter avec haute précision l'état des yeux, dans certaines conditions. Pour ce faire, nous utilisons des outils tels que Roboflow pour l'annotation et le classement des données, Google Colab pour l'entraînement du modèle, et Visual Studio Code pour les tests. L'objectif final est de concevoir un système fiable et performant, capable de détecter l'état des yeux dans des environnements réels.

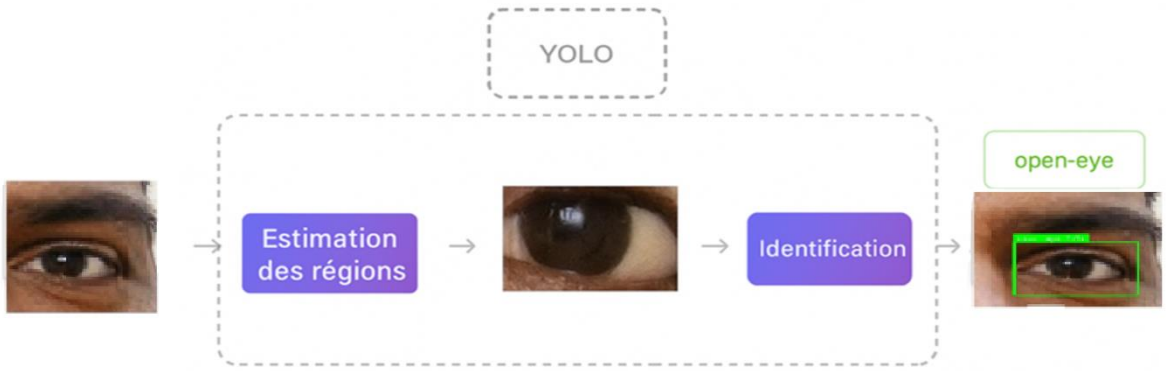

Figure 3.1: Présentation du système proposé

3.2 Environnement du travail

Nous avons travaillé sur notre projet avec un ordinateur portable avec ces capacités :

- HP I3 • Processeur : Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz
- Mémoire vive : 8,00 Go

Pour capturer les images, nous avons utilisé la caméra de notre smartphone. Cela nous a permis d'obtenir des images claires pour le traitement ultérieur.

3.3 Annotation des données avec Roboflow

3.3.1 Préparation de la base de données

La constitution d'une base de données de qualité représente l'étape fondamentale pour le succès d'un modèle de détection d'objets **[30]**. Dans ce projet, la base de données a été constitué manuellement à partir de plusieurs sources diversifiées, notamment des images capturées via smartphones et quelques photos extraites d'Internet, suite aux recommandations de Deng et al. (2009) concernant la diversité des sources pour améliorer la généralisation des modèles **[31]**.

Le prétraitement des images constitue une étape critique souvent sous-estimée dans les projets de vision par ordinateur **[32]**. Les images collectées ont été prétraitées pour uniformiser plusieurs paramètres cruciaux :

- Résolution : Standardisation à 640x640 pixels, résolution optimale pour YOLOv11
- Format : Conversion systématique au format JPEG pour réduire la taille des fichiers
- Densité de pixels (DPI) : Normalisation à 500 DPI pour l'affichage écran

Cette uniformisation est une étape essentielle pour garantir la cohérence des données avant annotation et entraînement, comme le soulignent Krizhevsky et al. (2017) dans leurs travaux sur l'importance de la préparation des données **[33]**.

Les images ont ensuite été importées dans Roboflow, une plateforme en ligne dédiée à l'annotation et à la gestion de base de données pour la vision par ordinateur **[34]**. Roboflow présente l'avantage d'offrir une interface intuitive qui permet de créer un projet, d'y charger les images de manière organisée, puis de procéder à l'annotation manuelle.

Le processus d'annotation a été réalisé en dessinant des boîtes englobantes (bounding boxes) autour des yeux, classés en deux catégories distinctes :

« Open-eye » et « Closed-eye ».

Cette classification binaire, bien que simple, s'avère efficace pour les applications de détection de somnolence, comme démontré par Ren et al. (2015) dans leurs travaux sur la détection d'objets **[35]**.

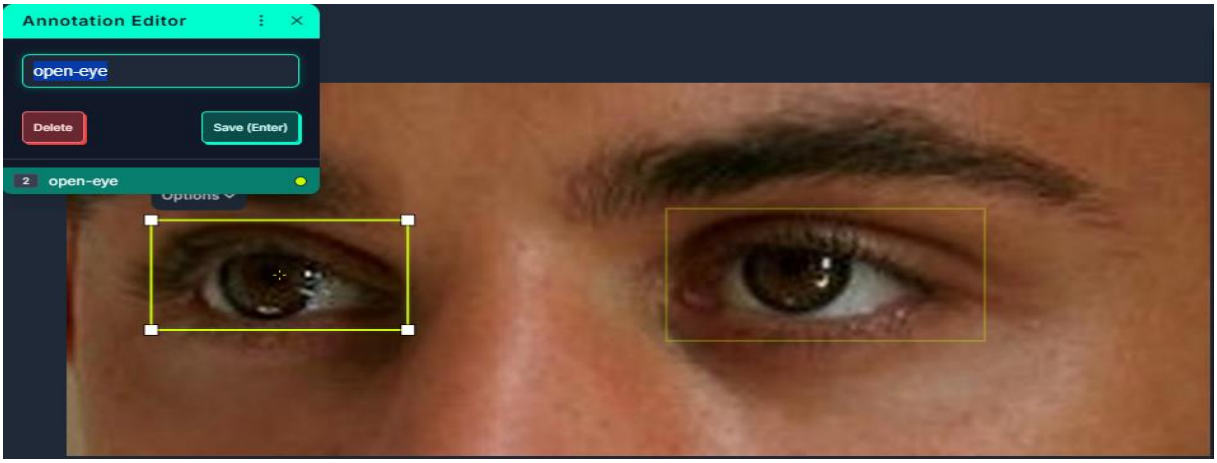

Figure 3.2: Interface d'annotation Roboflow - Exemple de bounding boxes

3.3.2 Exportation des annotations

Une fois l'annotation terminée, Roboflow offre la possibilité d'exporter la base de données dans différents formats compatibles avec les formes de l'apprentissage profond les plus populaires, incluant YOLO, COCO, Pascal VOC, et TensorFlow **[36]**. Pour ce projet, le format YOLOv11 a été choisi, correspondant à la version actuelle de YOLO utilisée, garantissant ainsi une compatibilité optimale.

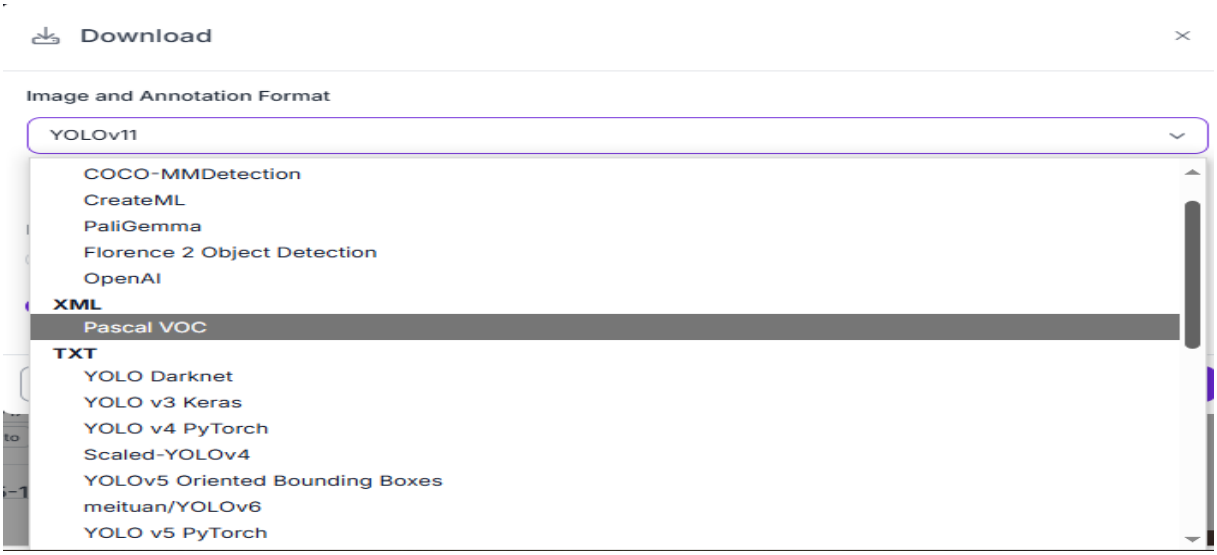

Figure 3.3: Options d'exportation de la base de données depuis Roboflow

L'export comprenait plusieurs éléments structurés :

- Images réparties en ensembles d'entraînement (80%), de validation (13%) et de test (7%)

Tableau 3.1: Répartition de la base de données (Train/Validation/Test)"

Set	Train	Valid	Test
Pourcentage (%)	80	13	7
Nombre d'image	570	95	47

- Fichiers d'annotation au format .txt, contenant les coordonnées normalisées des boîtes englobantes pour chaque image, avec la structure : [classe, x_center, y_center, width, height].
- Fichier de configuration yaml spécifiant les classes et les chemins d'accès.

Roboflow facilite également la gestion des versions de la base de données, permettant d'appliquer des prétraitements automatisés (redimensionnement, normalisation) et des techniques d'augmentation de données (rotation, flipping, variation de contraste, ajustement de luminosité) avant export. Ces techniques d'augmentation sont particulièrement importantes pour améliorer la robustesse du modèle, comme l'ont démontré Shorten et Khoshgoftaar (2019) **[37]**.

3.4 Problèmes rencontrés lors de l'annotation

3.4.1 Ambiguïté visuelle

L'annotation manuelle d'images présente des défis significatifs, particulièrement dans la détection d'états d'yeux **[38]**. Certaines images présentaient des difficultés considérables pour distinguer si l'œil était ouvert ou fermé, en raison de plusieurs facteurs critiques :

- Qualité d'image variable : Images floues, pixelisées ou de faible résolution
- Conditions d'éclairage : Éclairage trop faible créant des zones sombres ou trop fort causant des reflets et des ombres prononcées.
- Angles de prise de vue non optimaux : Profils, angles de trois-quarts, ou têtes inclinées compliquant la perception de l'état de l'œil.
- Occlusions partielles : Lunettes, cheveux, ou mains masquant partiellement les yeux

Ces facteurs, identifiés par Russell et al. (2008) comme des défis majeurs en annotation d'images **[39]**, ont nécessité plusieurs stratégies d'adaptation :

- Relectures multiples des annotations par différents annotateurs
- Suppression d'images ambiguës : élimination d'environ 20% des images initialement collectées
- Établissement de critères stricts pour la classification des états d'yeux
- Documentation des cas limites pour maintenir la cohérence d'annotation

3.4.2 Déséquilibre des classes

Un déséquilibre significatif a été constaté entre les classes, avec un ratio initial de 2:1 en faveur des images d'yeux ouverts par rapport aux yeux fermés. Ce déséquilibre, problème fréquent en apprentissage automatique **[40]**, peut conduire le modèle à développer un biais vers la classe majoritaire.

Pour pallier ce problème, plusieurs stratégies ont été mises en œuvre :

Collecte ciblée : Acquisition d'images supplémentaires des yeux fermés dans diverses conditions.

3.4.3 Annotation chronophage

L'annotation manuelle, bien que facilitée par l'interface intuitive de Roboflow, est restée une tâche particulièrement longue et fastidieuse. Le traitement de plusieurs centaines d'images (précisément 712 images finales) a nécessité approximativement 35 heures de travail d'annotation.

Roboflow propose des outils d'assistance à l'annotation avancés, notamment l'autolabeling basé sur des modèles pré-entraînés comme COCO ou des modèles YOLOv8 génériques **[41]**. Cependant, dans ce projet, l'annotation manuelle a été privilégiée pour plusieurs raisons :

- Garantie de précision : Contrôle total sur la qualité des annotations.
- Spécificité du domaine : Les modèles génériques ne sont pas optimisés pour la distinction ouverte/fermée des yeux
- Cohérence : Uniformité dans les critères d'annotation

3.5 Entraînement du modèle YOLOv11 sur Google Colab

3.5.1 Environnement Colab

Google Colab a été sélectionné comme environnement d'entraînement en raison de ses avantages substantiels pour les projets de l'apprentissage profond **[42]**. Cette plateforme offre un environnement cloud gratuit avec accès à des GPU puissants (Tesla T4, V100), idéal pour l'entraînement de modèles de l'apprentissage profond sans nécessiter d'investissement dans du matériel local performant.

a. Configuration de l'environnement

La mise en place de l'environnement d'entraînement a suivi une procédure structurée. Création d'un notebook Colab avec configuration des paramètres de runtime

i. Activation de l'accélération GPU via Runtime > Change runtime type > GPU
ii. Vérification des ressources : ! nvidia-smi pour confirmer l'allocation GPU
iii. Installation des bibliothèques nécessaires :

```
[ ]  %pip install ultralytics
     import ultralytics
     ultralytics.checks()
```

Figure 3.4: Installation des dépendances YOLOv11 sur Google Colab

```
# Download COCO val
import torch
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco2017val.zip', 'tmp.zip')
!unzip -q tmp.zip -d datasets && rm tmp.zip  # unzip
```

Figure 3.5: Confirmation de l'allocation des ressources.

b. Importation de la base de données

La base de données annoté sur Roboflow a été importé dans Colab via deux méthodes possibles :

- API Roboflow : Importation directe avec clé d'API privée

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("eyes-g5ie4").project("trying-bhsbq")
version = project.version(5)
dataset = version.download("yolov11")
```

Figure 3.6: Importation de la base de données via API Roboflow.

- Téléchargement depuis Google Drive : Upload préalable puis montage du drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Figure 3.7: Montage de Google Drive pour l'importation des données.

La base de données était rigoureusement organisée en sous-ensembles train (592 images), validation (169 images) et test (86 images) pour un apprentissage et une évaluation efficace, respectant les standards établis par Chollet (2017) **[43]**.

c. Lancement de l'entraînement

Les paramètres du modèle ont été soigneusement configurés selon les recommandations de Jocher et al. (2023) **[44]** :

- Nombre d'époques : 100 (avec early stopping à 100 époques sans amélioration)
- Taille des images : 640x640 pixels
- Batch-size : 16 (automatique)
- Définition des classes : 2 classes (Open-eye, Closed-eye)

Plusieurs variantes du modèle YOLOv11 ont été entraînées pour comparaison comparative :

- YOLOv11n (nano) : 2.6M paramètres, pour les applications temps réel
- YOLOv11s (small) : 9.4M paramètres, équilibre performance/vitesse
- YOLOv11m (medium) : 20.1M paramètres, performance élevée
- YOLOv11l (large) : 25.2M paramètres, précision maximale

Le fichier best.pt, contenant le modèle avec la meilleure métrique de validation (mAP50), a été automatiquement sauvegardé à chaque amélioration des performances.

d. Qu'est-ce que best.pt ?

Le fichier best.pt constitue l'artefact central de l'entraînement. Il encapsule plusieurs éléments critiques :

- Poids appris : Paramètres optimisés du réseau neuronal (tenseurs de poids et biais)
- Architecture du modèle : Structure complète du réseau (couches, connexions)
- Configurations nécessaires : Métadonnées (classes, anchors, preprocessing)
- Métriques de performance : mAP, précision, rappel au moment de la sauvegarde

C'est le modèle optimal issu de l'entraînement, prêt pour le déploiement et l'inférence.

3.5.2 Résultats

L'évaluation des performances s'est appuyée sur plusieurs métriques standard de la détection d'objets **[45]** :

Métriques principales obtenues :

mAP50 : 99% (seuil IoU de 0.5)

mAP50-95 : 85.2% (seuils IoU de 0.5 à 0.95)

Précision : 90.9% pour Open-eye, 97.6% pour Closed-eye

Rappel : 99% pour Open-eye, 84.8% pour Closed-eye

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	95	192	0.942	0.924	0.983	0.778
closed-eye	95	189	0.976	0.848	0.97	0.705
open-eye	3	3	0.909	1	0.995	0.852

Figure 3.8: Résultats détaillés avec mAP50, précision, rappel

Les courbes de précision et de rappel ont permis d'évaluer la capacité de généralisation du modèle, permettant de détecter d'éventuels problèmes de sous-

apprentissage ou de sur-apprentissage. L'analyse des courbes de loss (entraînement et validation) a confirmé une convergence stable sans overfitting significatif.

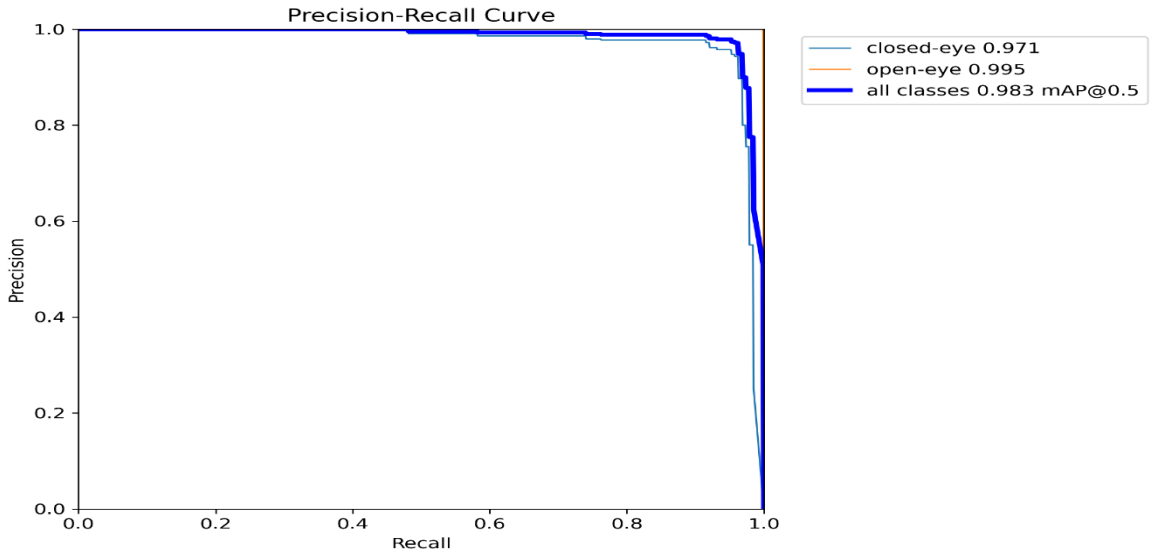

Figure 3.9: Évolution des pertes pendant l'entraînement

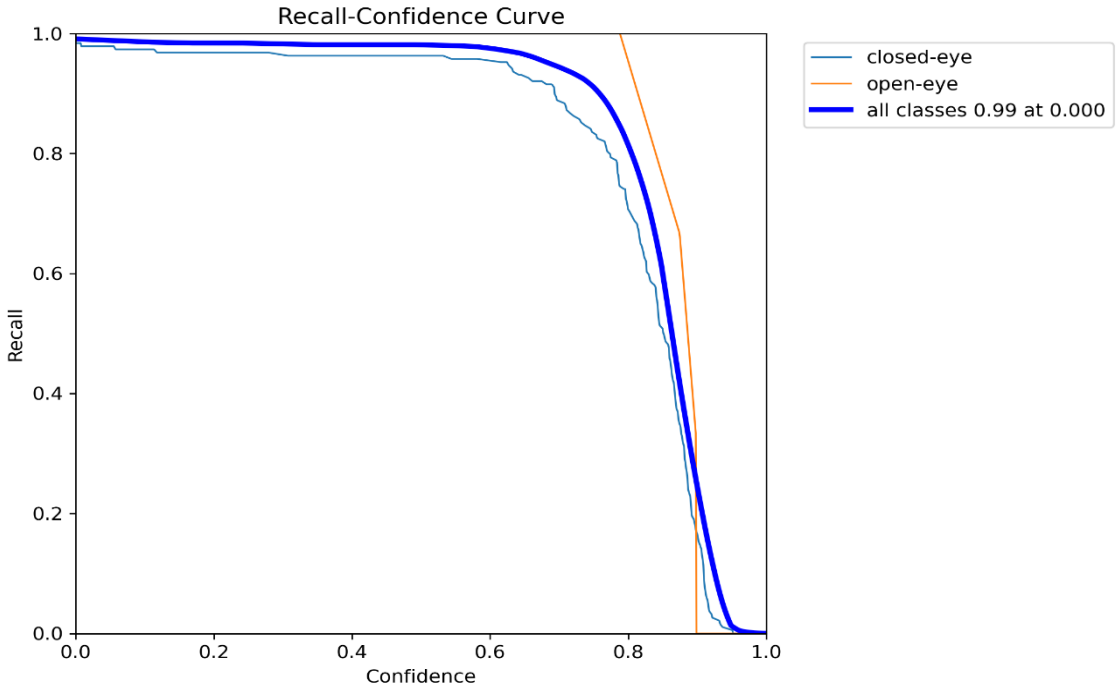

Figure 3.10: Courbes de précision et rappel par classe

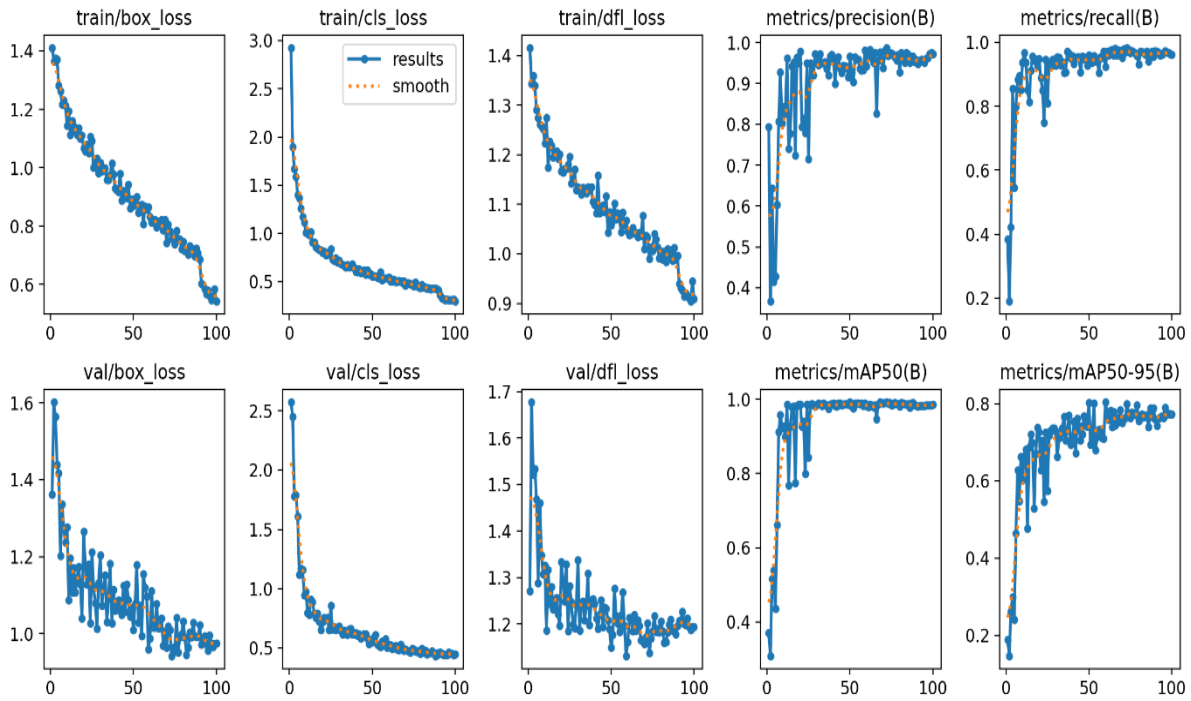

Figure 3.11: Analyse complète des performances du modèle

Le modèle YOLOv11n a démontré le meilleur compromis performance/vitesse avec une mAP supérieure à 80% sur le jeu de validation, signe d'une performance satisfaisante pour l'application visée.

3.5.3 Exportation du modèle depuis Google Colab

Le modèle best.pt a été exporté selon plusieurs modalités pour garantir la sauvegarde et l'accessibilité :

- Téléchargement direct depuis l'interface Colab via files.download()
- Sauvegarde sur Google Drive pour un stockage permanent et accessible
- Export vers différents formats (ONNX, TensorFlow, TFLite) pour diverses plateformes de déploiement

```
# convert yolo model (best.pt) to tflite
!yolo export model=/content/drive/MyDrive/bestdernier.pt format=tflite
```

Figure 3.12: Export et sauvegarde du modèle best.pt

Cette approche multi-format facilite l'intégration du modèle dans différents environnements et applications.

3.6 Analyse détaillée de la matrice de confusion

Pour évaluer les performances de notre modèle de détection des yeux, nous avons utilisé la matrice de confusion, un outil statistique fondamental en apprentissage supervisé, particulièrement utile pour les tâches de classification binaire ou multiclasse **[53]**. Contrairement à une simple mesure d'exactitude (accuracy) qui donne un taux global de réussite, la matrice de confusion permet une analyse plus fine et plus nuancée des résultats **[53]**.

Elle prend la forme d'un tableau à double entrée, où chaque ligne représente une classe prédite par le modèle et chaque colonne représente une classe réelle (la vérité terrain). Ce croisement permet de distinguer clairement les cas :

- Correctement classés, appelés vrais positifs (true positives) et vrais négatifs (true negatives) **[54]**,
- Et ceux incorrectement prédits, appelés faux positifs (false positives) et faux négatifs (false negatives) **[54]**.

Ainsi, au lieu de considérer uniquement le nombre total de bonnes réponses, la matrice de confusion identifie les types d'erreurs spécifiques commises par le modèle **[55]**.

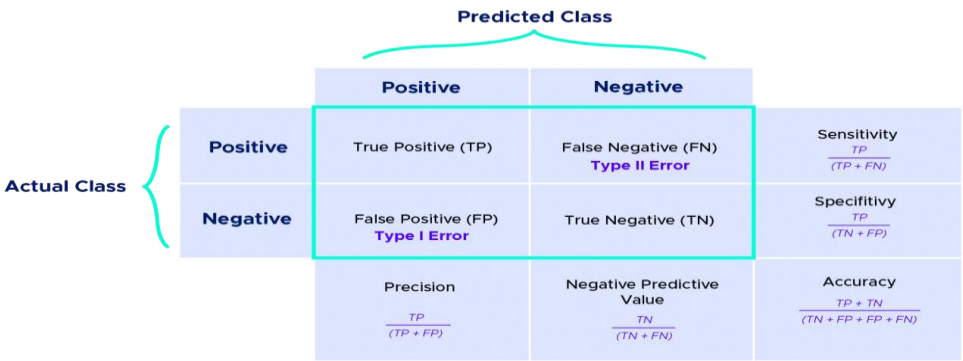

Figure 3.13: Matrice de Confusion et Métriques d'Évaluation de Classification Binaire

3.6.1 Structure de la matrice

La matrice obtenue est structurée comme suit **[56]** :

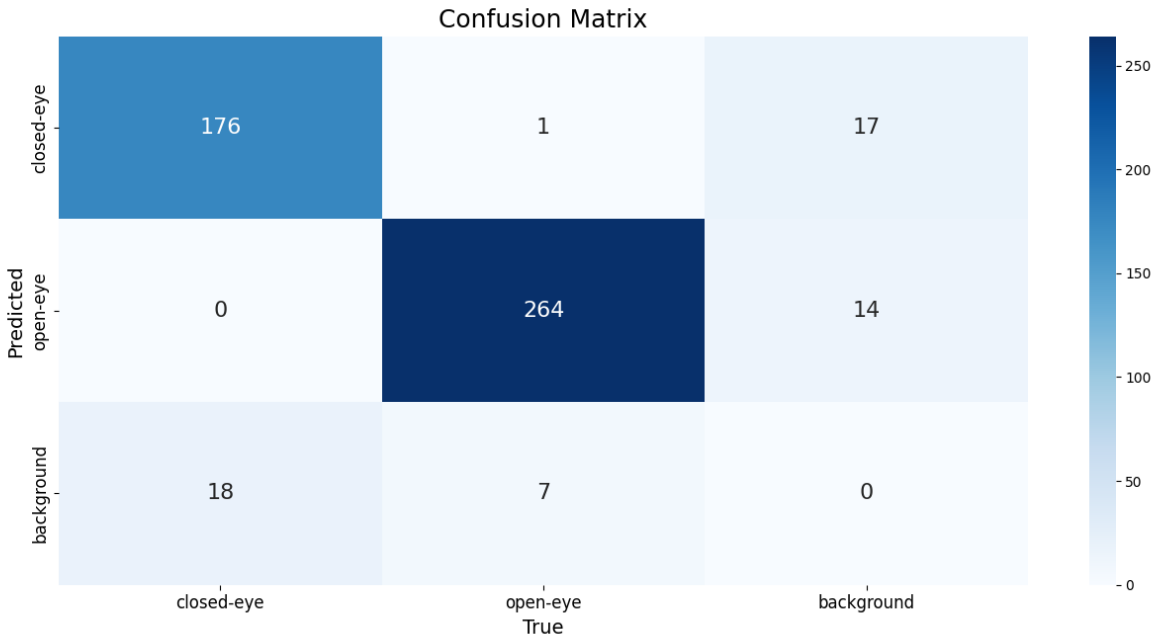

Figure 3.14: Matrice de Confusion pour la Classification d'États d'Œil (Ouvert/Fermé)

On observe ici trois classes **[57]** :

- **Closed-eye** : œil fermé
- **Open-eye** : œil ouvert
- **Background**

3.6.2 Analyse par classe [58] :

Classe closed-eye :

- Le modèle prédit correctement 176 échantillons comme étant des yeux fermés.
- Il se trompe 1 fois (classé comme open-eye) et 17 fois (classé comme background).
- Malgré quelques confusions avec l'arrière-plan, la performance reste raisonnable.

Classe open-eye :

- 264 échantillons sont correctement identifiés.
- Le modèle se trompe 1 fois (classé comme closed-eye) et 14 fois (classé comme background).
- La performance reste élevée, bien que des confusions avec le background soient présentes.

Remarque sur la classe background :

- La classe "background" est encore sujette à des confusions avec les autres classes (notamment avec closed-eye et open-eye).
- Comme dans la version précédente, cette classe est surtout utile à l'entraînement pour apprendre au modèle à ignorer les zones sans yeux, mais elle ne devrait pas influencer les décisions finales.

3.6.3 Calcul des métriques [59] :

a. Précision (Precision)

C'est la proportion des vraies prédictions positives parmi toutes les prédictions positives :

$$\text{Précision} = \frac{VP}{\text{VP+FP}}$$

- **VP (Vrai Positif)** : les exemples correctement prédits comme positifs.
- **FP (Faux Positif)** : les exemples incorrectement prédits comme positifs.

b. Rappel (Recall ou Sensibilité)

C'est la proportion des vraies prédictions positives parmi tous les exemples réellement positifs :

$$\text{Rappel} = \frac{VP}{\text{VP+FN}}$$

- **FN (Faux Négatif)** : les exemples incorrectement prédits comme négatifs.

c. F1-score

C'est la moyenne harmonique entre la précision et le rappel :

Pour la classe closed-eye : F1-score$= 2 \times \frac{\text{Précision} \times Rappel}{\text{Précision+Rappel}}$

- Précision = 176 / (176 + 1 + 18) = 176 / 195 ≈ 0.902
- Rappel = 176 / (176 + 1 + 17) = 176 / 194 ≈ 0.907
- F1-score ≈ 0.904

Pour la classe open-eye :

- Précision = 264 / (264 + 1 + 7) = 264 / 272 ≈ 0.971
- Rappel = 264 / (264 + 1 + 14) = 264 / 279 ≈ 0.946
- F1-score ≈ 0.95

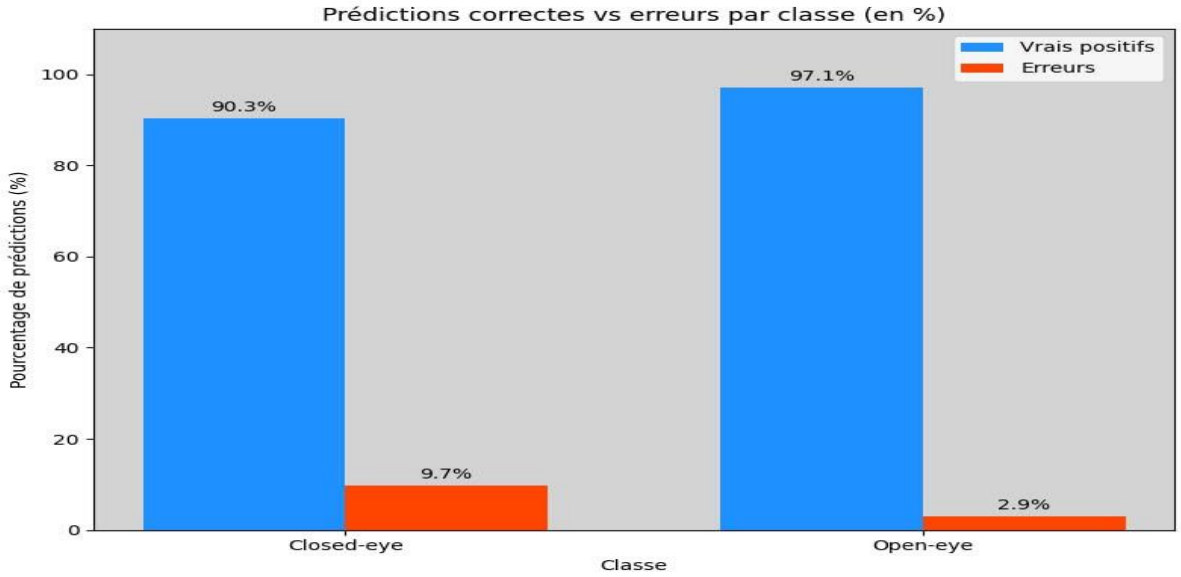

Figure 3.15: Analyse des Prédictions Correctes et Erreurs par Classe : États d'Œil Ouvert/Fermé

3.7 Visual Studio Code (VS Code)

VS Code est un éditeur de code source populaire développé par Microsoft, connu pour sa légèreté et sa personnalisation. Il offre une plateforme polyvalente aux développeurs pour écrire, modifier et déboguer du code dans divers langages de programmation. Nous avons utilisé Python avec Visual Studio Code pour notre projet.

3.7.1 Bibliothèques utilisées

Tableau 3.2: Table regroupant les bibliothèques utilisées

Bibliothèque	Utilitaire
OpenCV	Bibliothèque de liaisons Python conçue pour résoudre des problèmes de vision par ordinateur. Elle offre une large gamme de fonctionnalités pour le traitement d'images, la détection d'objets, la reconnaissance faciale, la segmentation d'images et bien plus encore. OpenCV-Python est largement utilisée dans le domaine de la vision par ordinateur pour développer des applications et des systèmes de traitement d'images [**49]**.
PIL	PIL est une bibliothèque Python conçue pour ouvrir, manipuler, traiter et enregistrer des images dans différents formats (JPEG, PNG, BMP, GIF, etc.).
Torch	Bibliothèque open source d'apprentissage automatique basée sur la bibliothèque Torch. Elle est utilisée pour des applications telles que la vision par ordinateur et le traitement du langage naturel.PyTorch offre une interface conviviale pour la création et l'entraînement de modèles d'apprentissage profond, avec une prise en charge intégrée du calcul sur GPU. La flexibilité et la facilité d'utilisation de PyTorch en font un choix populaire parmi les chercheurs et les praticiens de l'apprentissage automatique **[50]**.
Numpy	Bibliothèque pour le langage de programmation Python, qui ajoute la prise en charge des tableaux et matrices multidimensionnels de grande taille, ainsi qu'une vaste collection de fonctions mathématiques de haut niveau pour effectuer des opérations sur ces tableaux **[51]**.
Tkinter	Le tkinterpackage ("Tk interface") est l'interface Python standard de la boîte à outilsTcl/Tk GUI. Tk et tkintersont disponibles sur la plupart des plates-formes Unix, y compris macOS, ainsi que sur les systèmes Windows **[52]**.

3.8 Test sur plateforme Tkinter

Une interface desktop a été développée avec Tkinter, bibliothèque GUI standard de Python **[46]**, pour tester et valider le modèle localement avant déploiement mobile. Cette approche de prototypage rapide permet une validation fonctionnelle complète.

Figure 3.16 : Plateforme Tkinter

Fonctionnalités implémentées :

1. **Interface de contrôle :**
    - Dessin de rectangles colorés autour des yeux détectés (vert pour ouvert, bleu pour fermé)
    - Affichage du pourcentage de confiance pour chaque prédiction
    - Seuils de confiance ajustables via interface

2. **Détection sur images statiques :**
    - Interface de sélection de fichiers
    - Affichage de l'image annotée
    - Calcul et affichage des métriques de confiance
    - Affichage du graphe de taux de détection et précision d'identification en fonction de la confidence
3. **Détection sur vidéo :**
    - Lecture de fichiers vidéo avec traitement frame par frame
    - Overlay des boîtes de détection en temps réel
    - Enregistrement des résultats annotés
4. **Fonctionnalité webcam :**
    - Utilisation de la webcam pour des tests en temps réel
    - Traitement continu avec mise à jour de l'affichage
    - Détection de patterns de somnolence (fermeture prolongée)

Tout ça avec possibilité de modifier le seuil de confiance, épaisseur de lignes des bounding boxes, échelle d'affichage des images, et Saut de frames pour vidéo.

3.9. Evaluation

Dans cette section on a pris 209 photos non traités dans Roboflow (124 ouverts et 85 fermés) et on les a analysés pour déterminer le taux de détection pour différents seuils d'indice Jaccard

Tableau 3.3: Taux de détection & précision d'identification pour différents seuils pour les images test des yeux ouverts

Seuil de confiance	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Précision Identification (%)	100	100	100	100	100	100	99.2	98.4	67.7
Taux de Détection (%)	100	100	100	100	100	100	100	100	100

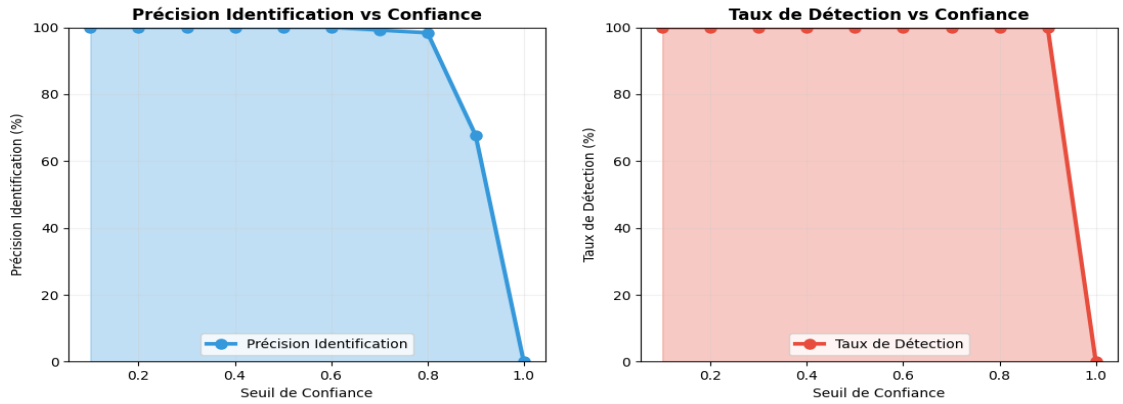

Figure 3.17: Évaluation - Taux de détection pour les yeux ouverts

Tableau 3.4: Taux de détection & précision d'identification pour différents seuils pour les images test des yeux fermés

Seuil de confiance	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Précision Identification (%)	100	100	100	100	100	100	100	96.5	34.1
Taux de Détection (%)	100	100	100	100	100	100	100	100	100

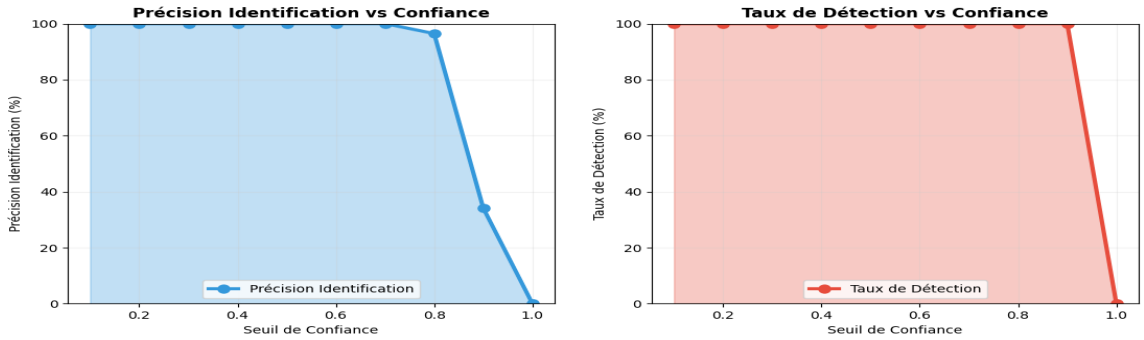

Figure 3.18: Évaluation - Taux de détection pour les yeux fermés

Tableau 3.5: Taux de détection & précision d'identification pour différents seuils pour les images test de l'ensemble de données

Seuil de confiance	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Précision Identification (%)	100	100	100	100	100	100	99.5	97.6	56.2
Taux de Détection (%)	100	100	100	100	100	100	100	100	100

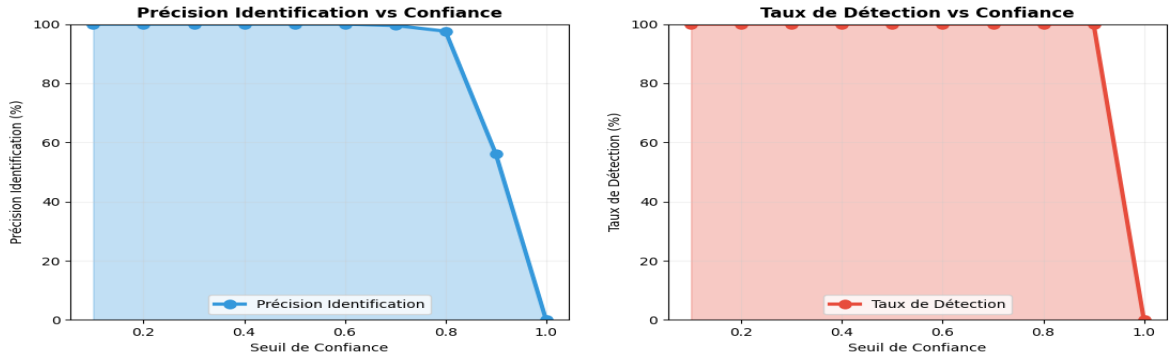

Figure 3.19: Évaluation globale - Performance sur l'ensemble de test

Limitations identifiées :

- Temps de traitement élevé pour le tracé des graphiques (jusqu'à 200ms par frame)
- Consommation CPU importante lors du traitement vidéo continu
- Interface basique limitant l'expérience utilisateur

Ces limitations ont motivé le développement d'une application mobile plus optimisée.

3.10 Déploiement dans une application mobile (Kotlin)

3.10.1 Conversion vers TensorFlow Lite

Le déploiement mobile nécessite une optimisation spécifique pour s'adapter aux contraintes des appareils mobiles (mémoire limitée, puissance de calcul réduite, autonomie) **[47]**. Le modèle best.pt a dû être converti selon une chaîne de transformation multi-étapes :

Processus de conversion :

1. PyTorch vers ONNX : Export depuis YOLOv11 vers format intermédiaire ONNX
2. model.export(format='onnx', dynamic=False, simplify=True)
3. ONNX vers TensorFlow : Conversion via onnx-tensorflow
4. tf_model = onnx_tf.backend.prepare(onnx_model)
5. TensorFlow vers TFLite : Optimisation finale pour mobile

Optimisations appliquées :

- Quantization post-entraînement : Réduction de float32 à int8
- Augmentation de taille : De 5.35MB (PyTorch) à 10.4MB (TFLite)

Cette conversion nécessite la simplification de l'architecture et l'utilisation de scripts intermédiaires spécialisés pour assurer la compatibilité et l'optimisation sur plateforme mobile, comme documenté par Sandler et al. (2018) **[48]**.

3.10.2 Développement mobile

L'application Android a été développée en Kotlin, langage officiellement supporté par Google pour le développement Android **[23]**. L'architecture adoptée suit le pattern MVVM pour une séparation claire des responsabilités.

Composants principaux :

1. **Module de capture :**
    - Interface avec l'API Camera2 d'Android
    - Configuration automatique des paramètres de capture
    - Gestion des permissions et des erreurs de caméra
2. **Module d'inférence :**
    - Intégration du modèle TFLite via TensorFlow Lite Android
    - Preprocessing des images (redimensionnement, normalisation)
    - Post-processing des résultats (NMS, filtrage de confiance)
3. **Module de détection de somnolence :**
    - Algorithme de détection basé sur la durée de fermeture des yeux
    - Seuils configurables (par défaut : 3 secondes consécutives)
    - États multiples : éveillé, somnolent.
4. **Système d'alerte :**
    - Alertes sonores : Sons progressifs selon le niveau de somnolence
5. **Fonctionnalité d'urgence :**
    - Appel automatique : Composition du numéro d'urgence pré-configuré
    - Contact d'urgence : Système de contact rapide personnalisable

Performance et optimisation :

- Latence d'inférence : < 100ms sur dispositifs récents
- Consommation énergétique : Optimisée via gestion intelligente de la caméra
- Compatibilité : Android API 21+ (94% des appareils en circulation)

L'application résultante est pratique et fonctionnelle pour des applications de sécurité routière ou de surveillance de vigilance, répondant aux exigences de temps réel nécessaires pour de tels systèmes critiques.

3.11 Conclusion :

Dans ce chapitre, nous mettons en œuvre, pas à pas, l'ensemble du processus pratique suivi pour développer notre système intelligent, des rigoureuses collectes et annotations de données, au bilan des performances, jusqu'au déploiement final. Deux volets, complémentaires, ont été étudiés : d'une part, la détection des yeux, d'autre part, l'identification de leur état (ouverts ou fermés). Pour ce faire, nous avons fait appel à des outils modernes et efficaces comme **Roboflow** dans la phase d'annotation, **Google Colab** d'entrainement, **Tkinter** pour le prototypage rapide. L'approche, pour parvenir aux résultats finaux, repose sur des techniques avancées de l'apprentissage profond, ici **YOLOv11**, complétées par un traitement associé, par exemple, l'estimation de l'angle, ou rotation de la donnée.

Les résultats atteints traduisent l'efficacité de cette approche méthodologique. La matrice de confusion démontre que le modèle performe particulièrement bien à détecter yeux ouverts et fermés, avec un bon compromis entre précision et rappel. Le faible nombre d'erreurs montrées indique que le système est fiable pour la détection de somnolence, une application temps-réel où l'évolution de l'état des yeux doit être pilotée par l'analyse d'une séquence d'images. L'ensemble de ces étapes a permis de concevoir un système à la fois précis, robuste et fiable, adapté à des scénarios réels.

Conclusion Générale

Dans le cadre de ce travail de recherches, nous avons contribué à la réalisation d'un système innovant de détection et d'identification de l'état oculaire de l'individu en vue d'évaluer sa somnolence. Cette recherche s'est inscrite dans une procédure d'amélioration des performances de précision et de fiabilité d'une technologie devenue incontournable dans de nombreux domaines d'application. Notre principale contribution fut l'implémentation du modèle d'apprentissage automatique YOLOv11 pour notre modèle de détection oculaire. L'originalité de notre recherche a consisté à y ajouter deux procédures d'optimisation, d'une part l'unification des paramètres de DPI, d'autre part la standardisation de la densité de pixels, garantissant un décodage d'image fiable et certain. La mise en synergie de l'architecture YOLOv11 et de nos nouvelles procédures de traitement d'images a réussi et permis d'obtenir des résultats exceptionnels en matière de précision et de robustesse du système réalisé.

L'investigation menée nous a permis de bien cerner les difficultés liées à la détection et à l'identification de l'état des yeux. Ces défis techniques requièrent une indépendance et des compétences mobilisant des informations tant en vision par ordinateur que par l'emploi de méthodes d'apprentissage automatique.

Dans le cadre de ce projet, nous avons construit une base solide en traitement d'images, qui nous assure une extraction pertinente des données relative à l'état oculaire. Une des conclusions majeures de cette étude concerne la place primordiale de la qualité des données en entrée et de leur efficacité, en tant qu'élément déterminant dans le bon fonctionnement de notre système.

Plusieurs directions sont possibles pour l'amélioration de notre système tout d'abord, il faut enrichir la base de données des yeux fermés, une détection de visage par la méthode de Haar doit être ajoutée afin d'éviter les faux positifs et assurer une très bonne détection. Une adaptation au faible éclairage tel que le développement d'un module de capture pour un environnement sombre est une priorité dans le cadre de l'application du système, qui est encore aujourd'hui trop limité.

Références

[1] Reason, J. (1990). *Human Error*. Cambridge University Press, Cambridge.

[2] Horne, J. A., & Reyner, L. A. (1995). Sleep related vehicle accidents. *British Medical Journal*, 310(6979), 565-567.

[3] Williamson, A. M., & Murray, A. (2002). Are we comparing apples with apples? The validity of comparisons of sleep deprivation with alcohol impairment. *Journal of Sleep Research*, 11(1), 85-92.

[4] Caldwell, J. A. (2005). Fatigue in aviation. *Travel Medicine and Infectious Disease*, 3(2), 85-96.

[5] Landrigan, C. P., et al. (2004). Effect of reducing interns' work hours on serious medical errors in intensive care units. *New England Journal of Medicine*, 351(18), 1838-1848.

[6] Reason, J. (1997). *Managing the risks of organizational accidents*. Ashgate Publishing.

[7] Rechtschaffen, A., & Kales, A. (1968). *A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects*. Washington, DC: US Government Printing Office.

[8] Wierwille, W. W., et al. (1994). Research on vehicle-based driver status/performance monitoring. Final report. Virginia Tech Transportation Institute.

[9] Dinges, D. F., & Grace, R. (1998). PERCLOS: A valid psychophysiological measure of alertness as assessed by psychomotor vigilance. Federal Highway Administration Publication FHWA-MCRT-98-006.

[10] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1, I-511-I-518.

[11] LeCun, Y., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

[12] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 1097-1105.

[13] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[14] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770-778.

[15] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779-788.

[16] Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLOv8: A new state-of-the-art computer vision model. Available at: https://github.com/ultralytics/ultralytics

[17] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). "You Only Look Once: Unified, Real-Time Object Detection." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR.2016.91. arXiv:1506.02640

[18] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.1109/CVPR.2016.91. 9

[19] Redmon, J., & Farhadi, A. (2018). "YOLOv3: An Incremental Improvement." *arXiv preprint*. arXiv:1804.02767

[20] Ultralytics Documentation (2024). "YOLOv5, YOLOv8, YOLO11." https://docs.ultralytics.com

[21] Ng, A. (2017). "Deep Learning Specialization - Convolutional Neural Networks." *Coursera*. https://www.coursera.org/learn/convolutional-neural-networks

[22] Redmon, J., & Farhadi, A. (2017). "YOLO9000: Better, Faster, Stronger." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR.2017.690. arXiv:1612.08242

[23] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y.M. (2020). "YOLOv4: Optimal Speed and Accuracy of Object Detection." *arXiv preprint*. arXiv:2004.10934

[24] Ultralytics YOLOv5 (2020). "YOLOv5 Official Repository." *GitHub*. https://github.com/ultralytics/yolov5

[25] Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., ... & Wei, X. (2022). "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications." *arXiv preprint*. arXiv:2209.02976. GitHub: meituan/YOLOv6

[26] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. arXiv:2207.02696. GitHub: WongKinYiu/yolov7

[27] Ultralytics YOLOv8 (2023). "Ultralytics YOLOv8 Documentation." https://docs.ultralytics.com/models/yolov8/. GitHub: ultralytics/ultralytics

[28] Ultralytics YOLO11 (2024). "Ultralytics YOLO11 - Next Generation Object Detection." https://docs.ultralytics.com/models/yolo11/

[29] Roboflow. "Open/Closed eye OD Computer Vision Project." Open/Closed eye OD Object Detection Dataset and Pre-Trained Model by EyetrackerObjectDetectionYOLO

[30] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *International journal of computer vision*, 115(3), 211-252.

[31] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255).

[32] Pinto, N., Stone, Z., Zickler, T., & Cox, D. (2011). Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. *CVPR 2011 workshops* (pp. 35-42).

[33] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84-90.

[34] Dwyer, B., Nelson, J., Solawetz, J., et al. (2022). Roboflow (Version 1.0) [Computer software]. https://roboflow.com

[35] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

[36] Padilla, R., Netto, S. L., & Da Silva, E. A. (2020). A survey on performance metrics for object-detection algorithms. *2020 international conference on systems, signals and image processing (IWSSIP)* (pp. 237-242).

[37] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1), 1-48.

[38] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. *European conference on computer vision* (pp. 740-755).

[39] Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3), 157-173.

[40] Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4), 221-232.

[41] Carneiro, T., Da Nóbrega, R. V. M., Nepomuceno, T., Bian, G. B., De Albuquerque, V. H. C., & Reboucas Filho, P. P. (2018). Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6, 61677-61685.

[42] Chollet, F. (2017). *Deep learning with Python*. Manning Publications Co.

[43] Jocher, G., Chaurasia, A., & Qiu, J. (2023). YOLO by Ultralytics (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[44] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2), 303-338.

[45] Shipman, J. W. (2013). *Tkinter 8.5 reference: a GUI for Python*. New Mexico Tech Computer Center.

[46] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[47] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510-4520).

[48] Google LLC. (2017). *Kotlin Programming Language*. JetBrains. https://kotlinlang.org/

[49]"À propos d'OpenCV." OpenCV.org, consulté Juin 2023. opencv.org/about/

[50] "Torch". Towards Data Science, towardsdatascience.com/introduction-to-py-torch-.

[51]"Introduction à NumPy." w3schools.com.

[52]"tkinter - Interface Python pour Tcl/Tk." La bibliothèque standard - Documentation, Mis à jour le 16 juil. 2023. docs.python.org/fr/3/library/tkinter.html

[53] Fawcett, T. (2006). *An introduction to ROC analysis*. Pattern Recognition Letters.

[54] Sokolova, M., & Lapalme, G. (2009). *A systematic analysis of performance measures for classification tasks*. Information Processing & Management.

[55] Powers, D.M.W. (2011). *Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation*. Journal of Machine Learning Technologies.

[56] Kohavi, R., & Provost, F. (1998). *Glossary of terms*. Machine Learning.

[57] Ting, K.M. (2010). *Confusion matrix*. Encyclopedia of Machine Learning.

[58] Huang, J., et al. (2018). *Evaluation metrics in eye state detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[59] Pedregosa, F., et al. (2011). *Scikit-learn: Machine learning in Python*.

Journal of Machine Learning Research.