# Développement Avancé : Reconnaissance d'Objets en Temps Réel
# Application Mobile de Détection d'Objets Scolaires

**Rapport de Projet - SAE 5.01 présenté par**

AIT BAHA Said, MORINON Lilian,
CHOLLET Thomas, KERBER Alexandre

**En vue de l'obtention du B.U.T. Informatique**

Académie de Nancy-Metz
Université de Lorraine
Institut Universitaire Technologique de Metz
Département Informatique
Promotion 2023-2026

## PARTIE 1 – INTRODUCTION & OBJECTIFS

Dans le cadre de la SAE 5.01 – Développement Avancé, notre groupe a réalisé une application mobile de reconnaissance d'objets du monde réel en temps réel, en s'appuyant sur des techniques modernes de vision par ordinateur et d'apprentissage profond. Cette SAE avait pour objectif de nous confronter à un projet combinant développement mobile et intelligence artificielle, tout en respectant des contraintes réalistes de performance, d'ergonomie et d'optimisation propres aux appareils mobiles.

L'objectif du projet n'était pas simplement de faire fonctionner une démonstration de reconnaissance d'objets, mais de concevoir une application réellement utilisable sur smartphone. Cela impliquait de prendre en compte les contraintes propres aux appareils mobiles, notamment la puissance de calcul limitée et la nécessité d'un affichage fluide en temps réel. L'application devait également proposer une interface claire et réactive afin de garantir une utilisation confortable pour l'utilisateur.

Nous avons choisi de concentrer notre travail sur la reconnaissance d'objets scolaires. Ce choix s'est imposé après une phase de réflexion sur la pertinence et la faisabilité du sujet. Les objets scolaires présentent l'avantage d'être des éléments du quotidien, facilement accessibles pour la constitution d'un jeu de données, tout en posant un véritable défi technique. En effet, plusieurs de ces objets sont visuellement très proches les uns des autres, notamment les stylos, les crayons et les surligneurs, qui partagent des formes et des proportions similaires. Cette forte similarité visuelle complexifie considérablement la tâche de classification et nécessite un modèle d'intelligence artificielle précis et bien entraîné.

Les objets que notre application est capable de reconnaître sont les suivants : la gomme, le surligneur, le tube de colle, le stylo, le crayon, la règle, les ciseaux, le taille-crayon et l'agrafeuse. Le choix de limiter le nombre de classes à ces neuf objets a été volontaire. Il visait à garantir une qualité de reconnaissance élevée, plutôt que de multiplier les classes au détriment de la précision, ce qui n'aurait pas été pertinent dans le cadre de cette SAE.

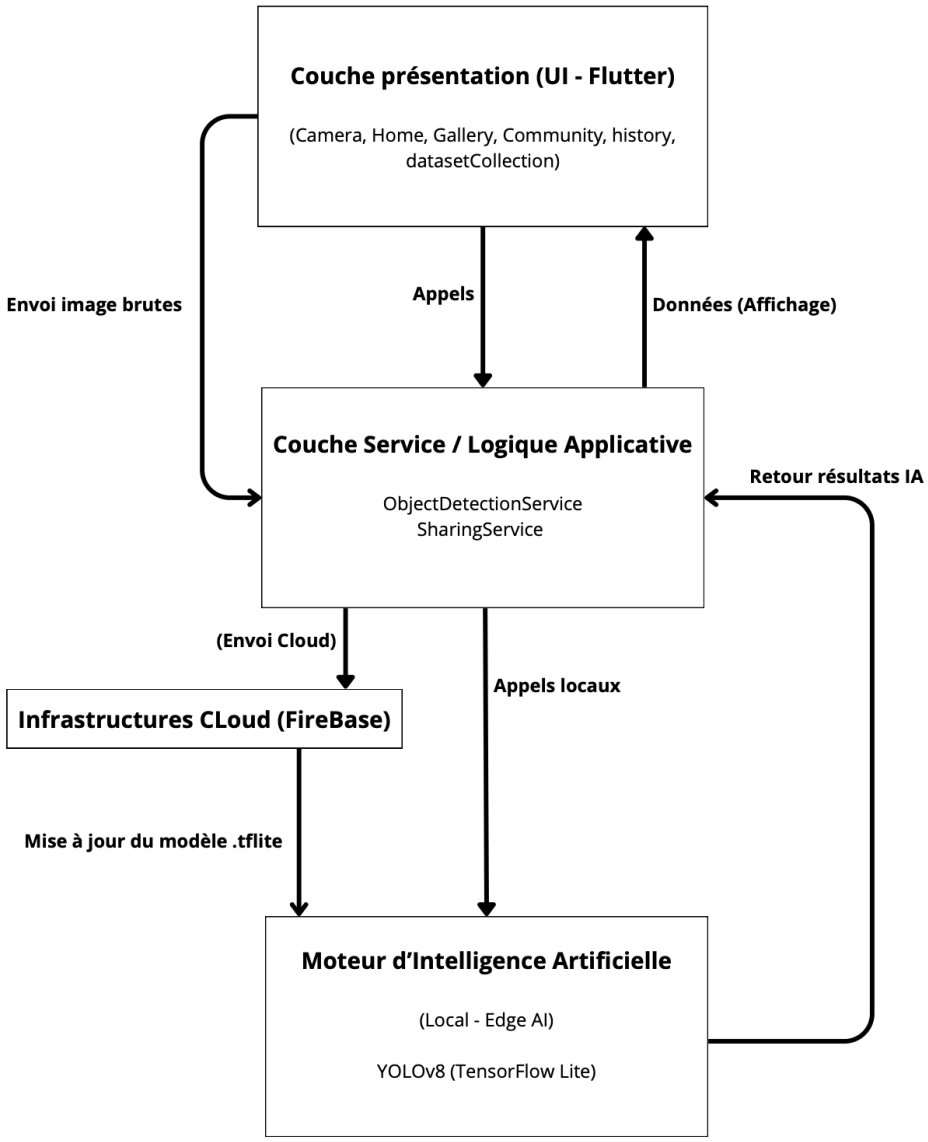
Ainsi, l'objectif global du projet était de concevoir une application mobile capable de détecter et de classifier ces objets scolaires en temps réel, avec un compromis maîtrisé entre précision et performance, tout en s'inscrivant dans une démarche d'ingénierie logicielle rigoureuse et cohérente avec les contraintes du développement mobile moderne.

## PARTIE 2 – ARCHITECTURE LOGICIELLE

L'architecture logicielle de l'application a été pensée afin de répondre à un objectif central : proposer une application mobile de reconnaissance d'objets réellement exploitable sur smartphone, et non une simple démonstration technique. Pour cela, il a été nécessaire de prendre en compte plusieurs contraintes propres au mobile, notamment les performances, la gestion de la caméra, l'exécution d'un modèle d'intelligence artificielle en temps réel et l'expérience utilisateur.

L'approche adoptée repose sur une architecture claire et modulaire, permettant de séparer les différentes responsabilités du projet. Cette organisation facilite la compréhension du fonctionnement global de l'application, améliore la maintenabilité du code et rend possible l'évolution future du projet sans remise en cause complète de l'existant.

**Architecture logicielle globale et flux de données du système**




### 2.1 Conception générale et choix technologiques

Le développement de l'application mobile a été réalisé à l'aide du framework Flutter, basé sur le langage Dart. Ce choix s'explique principalement par la capacité de Flutter à produire des applications performantes et multiplateformes à partir d'une base de code unique. Dans le cadre de ce projet, cette caractéristique était particulièrement pertinente, car l'application utilise intensivement la caméra du smartphone et nécessite un affichage fluide des résultats de détection en temps réel.

Flutter offre également une grande souplesse dans la conception des interfaces graphiques. Cela nous a permis de développer une application avec une interface moderne, réactive et cohérente, tout en conservant un bon niveau de lisibilité du code. La gestion de la navigation entre les écrans, des états de l'application et des interactions utilisateur s'intègre naturellement dans ce framework.

Pour la partie intelligence artificielle, nous avons fait le choix d'une exécution embarquée directement sur l'appareil mobile, via TensorFlow Lite. Cette solution permet d'exécuter un modèle de deep learning optimisé pour les environnements contraints, sans dépendre d'un serveur externe. Ce choix est déterminant pour garantir une faible latence, un fonctionnement hors ligne et une meilleure protection des données utilisateur.

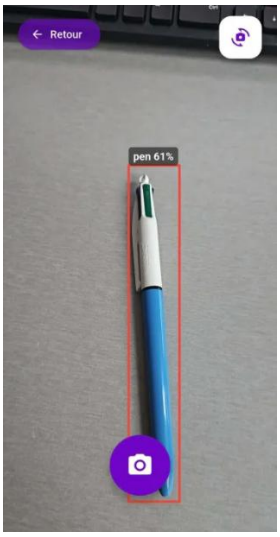
### 2.2 Organisation interne de l'application Flutter

L'application est structurée autour d'une organisation par écrans fonctionnels, chacun correspondant à une fonctionnalité précise. Cette organisation permet de découpler clairement les différentes parties de l'application et de limiter les dépendances inutiles entre les composants.

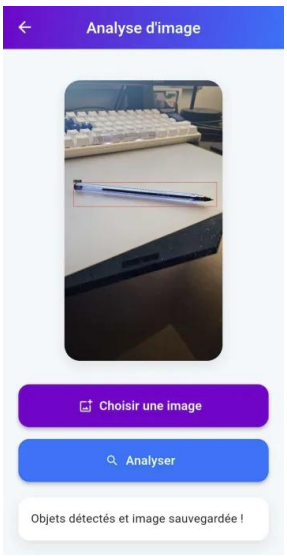
On distingue notamment :

- Un écran d'accueil, qui constitue le point d'entrée de l'application et permet d'accéder aux différentes fonctionnalités.

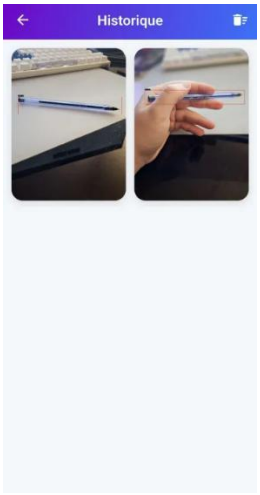

- Un écran de détection en temps réel, utilisant la caméra du smartphone pour analyser en continu le flux vidéo.



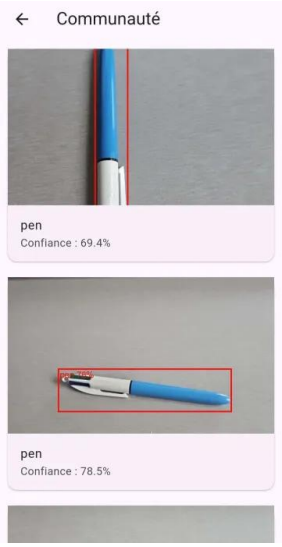

- Un écran d'analyse d'images depuis la galerie, permettant de traiter des images déjà enregistrées sur l'appareil.


- Un écran d'historique, qui affiche les images analysées et sauvegardées localement.



- Un écran communautaire, permettant de consulter les détections partagées par d'autres utilisateurs.




Cette séparation par écrans facilite la lecture du projet et permet d'isoler les comportements spécifiques à chaque fonctionnalité, tout en conservant une architecture globale cohérente.

### 2.3 Gestion de l'intelligence artificielle embarquée

La partie reconnaissance d'objets de l'application repose sur un module spécifique dédié à l'intelligence artificielle. Ce module regroupe toute la logique liée au chargement du modèle, à son exécution et au traitement des images fournies par l'application, que ce soit à partir du flux vidéo de la caméra ou d'images sélectionnées depuis la galerie.

Le modèle de détection est intégré directement dans l'application sous la forme d'un fichier model.tflite, accompagné d'un fichier de labels permettant d'associer les résultats du modèle aux différents objets scolaires reconnus. Le chargement du modèle est effectué au démarrage de l'application afin de limiter les temps d'attente lors de l'utilisation et d'assurer une exécution fluide dès l'ouverture des fonctionnalités de détection.

L'architecture mise en place permet également de remplacer le modèle utilisé, notamment pour intégrer un nouveau modèle issu d'une phase de ré-entraînement. Ce choix rend l'application plus évolutive, puisqu'il est possible d'améliorer les performances de détection ou d'ajouter de nouveaux objets sans modifier l'ensemble du code de l'application.

Deux modes d'analyse sont proposés à l'utilisateur :

- L'analyse en temps réel du flux vidéo de la caméra ;
- L'analyse d'images statiques provenant de la galerie du smartphone.

Dans les deux cas, l'exécution du modèle est réalisée localement sur le smartphone, sans communication réseau. Cette approche permet à l'application de fonctionner hors ligne et garantit une meilleure réactivité, tout en évitant l'envoi d'images vers un serveur distant.

Cependant, l'analyse en temps réel du flux caméra représente une charge importante pour le smartphone. Afin d'éviter une surcharge du processeur et d'assurer la stabilité de l'application, une régulation volontaire du nombre d'images analysées a été mise en place. Concrètement, une latence d'environ 300 millisecondes est introduite entre deux traitements successifs, ce qui limite le rythme d'analyse à environ 5 images par seconde.

Ce compromis permet de préserver les performances du smartphone et d'éviter des problèmes de stabilité lors d'une utilisation prolongée, tout en conservant une détection suffisamment réactive pour l'utilisateur. Ce choix s'inscrit dans une démarche réaliste, adaptée aux contraintes du développement mobile, et cohérente avec l'objectif de proposer une application réellement utilisable sur smartphone.

### 2.4 Chaîne de traitement des images

Le fonctionnement global de l'application repose sur une chaîne de traitement continue et structurée. Lorsqu'une image est capturée via la caméra ou sélectionnée depuis la galerie, celle-ci est d'abord prétraitée afin de correspondre au format attendu par le modèle d'intelligence artificielle. Ce prétraitement inclut notamment la gestion de l'orientation de l'image et la conversion vers un format exploitable par le moteur d'inférence.

L'image est ensuite transmise au moteur TensorFlow Lite, qui effectue la détection des objets scolaires présents. Pour chaque objet détecté, le modèle renvoie une classe, un score de confiance et les coordonnées d'une boîte englobante.

Ces informations sont alors exploitées par l'application Flutter pour afficher les résultats à l'écran, sous la forme de cadres positionnés dynamiquement sur l'image. Cette visualisation permet à l'utilisateur d'identifier immédiatement les objets détectés et d'évaluer la fiabilité de la reconnaissance.

### 2.5 Gestion du stockage local et des fonctionnalités sociales

Les images analysées peuvent être sauvegardées localement sur l'appareil, accompagnées des annotations générées par le modèle. Un mécanisme de stockage léger est utilisé pour conserver l'historique des analyses, permettant à l'utilisateur de consulter ultérieurement les résultats obtenus.

En complément, l'application propose une fonctionnalité de partage communautaire. Lorsqu'un utilisateur choisit de partager une détection, l'image et les informations associées sont envoyées vers une infrastructure distante reposant sur Firebase. Les images sont stockées via Firebase Storage, tandis que les métadonnées (label, score de confiance, date) sont enregistrées dans une base Firestore. Cette architecture permet d'afficher un flux communautaire mis à jour en temps réel, sans impacter le fonctionnement principal de l'application.

### 2.6 Justification globale des choix architecturaux

Les choix architecturaux réalisés tout au long du projet répondent directement aux contraintes du développement mobile moderne. L'exécution embarquée du modèle d'intelligence artificielle permet de

garantir une faible latence, un fonctionnement hors ligne et une meilleure protection des données utilisateur.

Le découpage de l’application en écrans fonctionnels et en modules dédiés facilite la lisibilité du projet et permet de séparer clairement l’interface utilisateur, la logique applicative et le moteur de détection. Cette organisation rend l’application plus maintenable et plus évolutive.

Enfin, l’architecture adoptée permet d’envisager facilement des évolutions futures, telles que l’ajout de nouveaux objets à détecter, l’amélioration du modèle ou l’enrichissement des fonctionnalités communautaires, sans remettre en cause la structure globale du projet.

# PARTIE 3 – CONSTITUTION DU DATASET ET PRÉPARATION DES DONNÉES

## 3.1 Problématique des données et contraintes liées aux objets scolaires

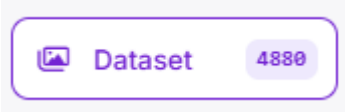
Les performances d'un modèle de reconnaissance d'objets dépendent fortement de la qualité et de la diversité des données utilisées lors de son entraînement. Dans le cadre de ce projet, la constitution du dataset a représenté une étape centrale et particulièrement exigeante, tant en termes de temps que de réflexion. En effet, le choix des objets à reconnaître et le contexte d'utilisation mobile imposaient des contraintes importantes sur la nature des données à collecter.

Les objets scolaires ciblés par notre application présentent une forte similarité visuelle. Certains objets ont des formes et des proportions très proches, ce qui rend leur distinction plus difficile pour un modèle d'intelligence artificielle. Cette difficulté est accentuée par le fait que ces objets sont souvent de petite taille et peuvent apparaître partiellement dans le champ de la caméra. Dans des scènes réelles, comme un bureau ou une table de classe, ils sont également entourés d'autres éléments visuellement perturbateurs.

Ces contraintes nous ont rapidement amenés à comprendre que la simple accumulation d'images ne suffirait pas. Il était nécessaire de construire un dataset varié, équilibré et représentatif des conditions réelles d'utilisation de l'application, afin de permettre au modèle d'apprendre à différencier correctement des objets pourtant très proches visuellement.

## 3.2 Collecte des données et constitution du dataset

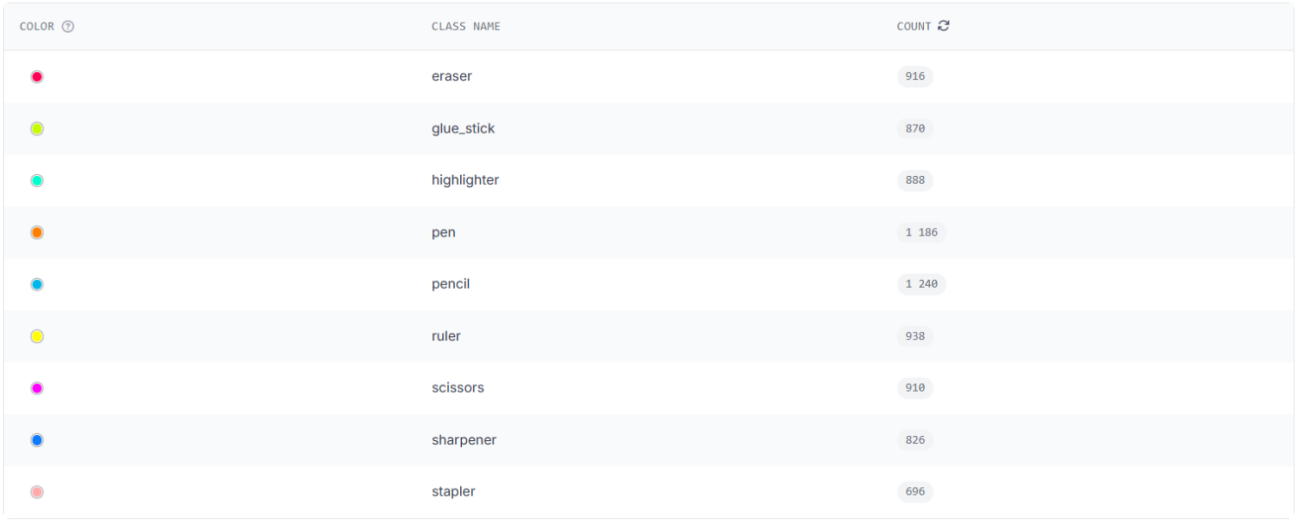
Le dataset final utilisé pour l'entraînement du modèle est composé de 4 880 images, réparties sur neuf classes correspondant aux objets scolaires sélectionnés. Nous avons volontairement limité le nombre de classes afin de conserver un bon niveau de précision pour chaque objet, plutôt que d'augmenter le nombre de catégories au détriment de la qualité de détection.




Afin d'améliorer la robustesse du modèle, nous avons également intégré des images dites de classe "null", c'est-à-dire des images ne contenant aucun des objets à reconnaître. Ces images peuvent représenter un bureau vide, une pièce entière, un mur ou encore des objets non scolaires. L'objectif de cette classe est d'apprendre explicitement au modèle qu'il n'est pas systématiquement nécessaire de détecter un objet dans une image. Cette approche permet de réduire de manière significative les faux positifs, en particulier dans des environnements complexes ou peu structurés.

La constitution du dataset s'est appuyée sur deux sources principales. Dans un premier temps, nous avons réalisé nos propres prises de vue d'objets scolaires, afin de disposer d'images réalistes, directement issues des conditions d'utilisation de l'application mobile. Cette méthode permettait de contrôler précisément les objets, les angles de vue et les situations rencontrées. Toutefois, elle s'est rapidement révélée insuffisante en termes de volume et de diversité, notamment pour couvrir un large éventail de conditions d'éclairage et d'arrière-plans.

Pour compléter ces données, nous avons utilisé des jeux de données open-source disponibles sur la plateforme Roboflow Universe. Cette plateforme nous a permis d'accéder à un grand nombre d'images déjà annotées, tout en conservant la possibilité de modifier ou de corriger les annotations existantes. Le recours à Roboflow a ainsi permis d'augmenter significativement la taille du dataset tout en maintenant un bon niveau de contrôle sur la qualité des données.



| COLOR | CLASS NAME | COUNT |
|---|---|---|
| | eraser | 916 |
| | glue_stick | 870 |
| | highlighter | 888 |
| | pen | 1 186 |
| | pencil | 1 240 |
| | ruler | 938 |
| | scissors | 910 |
| | sharpener | 826 |
| | stapler | 696 |

### 3.3 Nettoyage, annotation et structuration des données

Une fois les données collectées, un travail de vérification et de nettoyage du dataset a été réalisé afin de s'assurer que les images utilisées pour l'entraînement correspondaient bien aux objectifs du projet. Cette étape était importante pour garantir un apprentissage cohérent et éviter d'introduire des situations pouvant perturber le modèle.

Lors de cette phase, nous n'avons pas constaté de cas d'objets incorrectement étiquetés. En revanche, un problème fréquent concernait la présence d'objets scolaires non annotés dans certaines images. Par exemple, certaines images contenaient un taille-crayon, une gomme ou une règle correctement annotée, mais également un autre objet de notre liste posé à côté, non étiqueté. Cette situation s'explique par le fait que certains jeux de données utilisés à l'origine n'étaient pas conçus pour détecter l'ensemble des objets scolaires ciblés par notre application.

La présence de ces objets non annotés pouvait poser problème lors de l'entraînement, car le modèle n'était pas censé ignorer des objets qu'il doit ensuite apprendre à reconnaître. Afin de préserver la cohérence du dataset et d'éviter toute confusion lors de l'apprentissage, les images présentant ce type de situation ont été écartées lorsque cela était nécessaire.

En complément des images contenant un seul objet, nous avons également conservé des images présentant plusieurs objets simultanément, parfois de classes différentes. Cela permet au modèle de comparer les objets entre eux et de mieux comprendre ce qui distingue visuellement une classe d'une autre, notamment lorsque plusieurs objets similaires apparaissent dans la même scène. La présence de plusieurs objets dans une même image aide également le modèle à mieux appréhender leurs proportions et leur position relative dans la scène.

Enfin, l’ensemble du dataset a été structuré selon le format attendu par l’architecture YOLO, chaque image étant associée à un fichier d’annotation décrivant la classe de l’objet ainsi que les coordonnées de sa boîte englobante. Cette organisation permet une utilisation directe des données lors de l’entraînement et garantit un processus d’apprentissage clair et cohérent.

# PARTIE 4 – CHOIX DU MODÈLE DE DÉTECTION ET PHASE D'ENTRAÎNEMENT

Une fois le jeu de données préparé, nous avons dû choisir un modèle de détection adapté au projet. Cette partie présente le modèle retenu, les raisons de ce choix, ainsi que la manière dont il a été entraîné. Les résultats obtenus et les étapes nécessaires pour utiliser le modèle dans l'application mobile y sont également abordés.

## 4.1 Justification du choix du modèle de détection

Une fois le dataset constitué et correctement préparé, le choix du modèle de détection d'objets est devenu une étape clé du projet. L'objectif n'était pas uniquement d'obtenir de bons résultats théoriques, mais surtout de sélectionner un modèle capable de fonctionner efficacement dans une application mobile, avec des contraintes réelles de performance et de réactivité.

Le modèle devait répondre à plusieurs exigences :

- Être suffisamment précis pour distinguer des objets scolaires très proches visuellement (stylo, crayon, surligneur, etc.) ;
- Être assez rapide pour permettre une utilisation en temps réel via la caméra du smartphone ;
- Être compatible avec une exécution embarquée sur mobile, notamment via TensorFlow Lite.

Après avoir étudié différentes architectures de détection d'objets, nous avons choisi d'utiliser un modèle basé sur l'architecture **YOLO (You Only Look Once)**, et plus précisément une version récente adaptée à notre cas d'usage. YOLO appartient à la famille des modèles dits *single-stage*, ce qui signifie qu'il effectue la détection des objets en une seule étape, sans passer par une phase intermédiaire de proposition de régions.

Contrairement à des architectures *two-stage* comme Faster R-CNN, qui sont souvent plus lourdes et plus lentes, YOLO se distingue par une latence beaucoup plus faible. Cette caractéristique est essentielle dans le cadre d'une application mobile, où l'utilisateur s'attend à voir les résultats apparaître instantanément à l'écran. Le compromis proposé par YOLO entre vitesse d'exécution et précision correspondait parfaitement aux besoins du projet.

De plus, le fait que YOLO analyse l'image dans son ensemble permet au modèle de mieux prendre en compte le contexte global de la scène. Cela réduit les erreurs liées à l'arrière-plan et améliore la cohérence des détections, notamment lorsque plusieurs objets scolaires sont présents simultanément dans l'image.

## 4.2 Phase d'entraînement et paramètres du modèle

L'entraînement du modèle a été réalisé à partir du dataset préparé précédemment, en utilisant des environnements de calcul adaptés tels que **Kaggle** et **Google Colab**. Ces plateformes offrent un accès à des GPU NVIDIA T4, indispensables pour entraîner un modèle de détection sur un volume important d'images dans des délais raisonnables.

Le modèle a été entraîné sur un total de **148 epochs**. Ce nombre relativement élevé a été choisi afin de permettre au réseau de converger correctement et d'apprendre des caractéristiques suffisamment fines pour distinguer des objets très similaires visuellement. Cette phase a été particulièrement importante

pour les classes présentant des ressemblances visuelles, qui peuvent être confondues si l'entraînement n'est pas suffisant.

Les images du dataset ont été redimensionnées à une résolution de **960 × 960 pixels** avant l'entraînement, afin d'obtenir un format carré adapté au modèle de détection. Ce choix permet de conserver un bon niveau de détail, notamment pour les petits objets, tout en restant compatible avec une exécution future sur mobile après conversion du modèle. Bien que l'augmentation de la résolution entraîne un temps d'entraînement plus long et une consommation de ressources plus importante, ce compromis a été assumé afin de privilégier la qualité des détections.

Afin d'améliorer la robustesse du modèle, des techniques d'augmentation de données intégrées à l'architecture YOLO ont également été utilisées, en particulier la technique de **mosaïque**. Cette méthode consiste à combiner plusieurs images au sein d'une même entrée lors de l'entraînement. Elle permet d'augmenter la diversité des situations rencontrées par le modèle et d'améliorer sa capacité à généraliser à des scènes complexes, comme un bureau contenant plusieurs objets scolaires.

### 4.3 Évaluation des performances et conversion du modèle

Les performances du modèle entraîné ont été évaluées à l'aide de métriques standard utilisées en détection d'objets. Le modèle obtient un score de **mAP50 de 89,8 %**, ce qui indique une très bonne capacité à détecter et localiser correctement les objets présents dans les images. Le score **mAP50–95 de 77,3 %** montre quant à lui que le positionnement des boîtes englobantes reste précis sur une large plage de seuils d'intersection.

Ces résultats confirment que le modèle est capable de distinguer efficacement les différents objets scolaires, tout en conservant une précision suffisante pour une utilisation en temps réel dans une application mobile. Les performances obtenues sont cohérentes avec les objectifs initiaux du projet, qui privilégiaient la fiabilité et la réactivité plutôt qu'une précision extrême au détriment de la vitesse.

Une fois l'entraînement terminé, le modèle a été exporté puis converti vers le format **TensorFlow Lite**, étape indispensable pour permettre son exécution directement sur les appareils mobiles. Cette conversion a nécessité certaines adaptations, notamment en raison des différences de format de sortie entre le modèle d'origine et le moteur d'inférence embarqué. Ces ajustements ont permis d'assurer une intégration fluide du modèle dans l'application Flutter et une exploitation correcte des résultats lors de l'affichage des détections en temps réel.

Ainsi, le choix du modèle YOLO, associé à un entraînement approfondi et à un dataset soigneusement préparé, a permis de mettre en place un moteur de détection performant, fiable et adapté aux contraintes d'une application mobile moderne.

# PARTIE 5 – INTÉGRATION DU MODÈLE SUR MOBILE ET OPTIMISATIONS

## 5.1 Choix d'une exécution embarquée et intégration de TensorFlow Lite

Une fois le modèle de détection entraîné et validé, l'étape suivante a consisté à l'intégrer directement au sein de l'application mobile. Cette phase a nécessité de prendre en compte les contraintes spécifiques aux smartphones, qui disposent de ressources limitées par rapport à un ordinateur ou à un serveur dédié. Il ne s'agissait donc pas seulement de faire fonctionner le modèle, mais de garantir une utilisation fluide, réactive et adaptée à un usage réel.

Contrairement à une architecture reposant sur un serveur distant, nous avons fait le choix d'une exécution entièrement embarquée du modèle sur le smartphone. Cette décision permet d'éviter toute dépendance à une connexion réseau, de réduire fortement la latence et d'améliorer la confidentialité des données. Les images analysées par l'application ne quittent jamais l'appareil, ce qui représente un avantage important du point de vue de la protection des données utilisateur.

Pour rendre cette exécution possible sur mobile, le modèle a été converti au format TensorFlow Lite. TensorFlow Lite est une version allégée de TensorFlow, spécialement conçue pour les environnements embarqués comme les smartphones. Elle permet d'exécuter des modèles d'intelligence artificielle directement sur le processeur ou le GPU de l'appareil, tout en limitant l'impact sur les performances globales du système.

Cette solution s'est imposée comme la plus cohérente avec les objectifs du projet, qui visaient à proposer une application autonome, utilisable en toutes circonstances et capable de fournir des résultats en temps réel.

Plusieurs autres solutions auraient pu être envisagées pour l'exécution du modèle d'intelligence artificielle sur mobile. Par exemple, PyTorch Mobile permet également d'exécuter des modèles directement sur un appareil. Cependant, cette solution est généralement moins intégrée à l'écosystème Android et nécessite davantage de configuration. À l'inverse, TensorFlow Lite offre une intégration plus simple et plus stable pour les applications mobiles, en particulier dans un contexte Android et avec un framework multiplateforme comme Flutter.

La solution CoreML aurait également pu être envisagée, mais elle est exclusivement destinée aux appareils Apple fonctionnant sous iOS. L'application ayant été développée pour Android, ce choix n'était pas adapté au contexte du projet et aurait fortement limité la compatibilité de l'application.
Enfin, des solutions comme ONNX Runtime sont principalement utilisées pour l'exécution de modèles sur des environnements web ou sur des ordinateurs. Bien qu'efficaces dans ces contextes, elles sont moins adaptées à une exécution embarquée sur smartphone, notamment en raison des contraintes de performances et d'intégration sur mobile.

Ainsi, TensorFlow Lite s'est imposé comme la solution la plus cohérente pour notre projet, car elle est spécifiquement conçue pour l'exécution de modèles sur des appareils mobiles, offre de bonnes performances, une intégration native avec Android et une compatibilité directe avec notre chaîne d'entraînement.

## 5.2 Optimisations techniques et gestion des performances

L'intégration du modèle TensorFlow Lite dans l'application Flutter a nécessité la mise en place d'une stratégie de chargement efficace. Le modèle est stocké directement dans les ressources de l'application

et chargé une seule fois au démarrage. Cette approche permet d'éviter des temps d'attente répétés lors de l'analyse des images et d'assurer une meilleure réactivité lors de l'utilisation de la caméra ou de la galerie.

Un travail important a également été réalisé sur le prétraitement des images. Les images fournies par la caméra du smartphone sont généralement disponibles dans des formats spécifiques, tels que YUV ou BGRA, qui ne correspondent pas directement au format attendu par le modèle. Il a donc été nécessaire de convertir ces images tout en conservant un maximum de qualité, afin de ne pas dégrader les performances de détection. Cette étape est essentielle, car une conversion mal maîtrisée peut entraîner des erreurs de reconnaissance ou une baisse de précision.

Concernant l'optimisation du modèle, nous avons choisi de conserver le format **float32** lors de la conversion vers TensorFlow Lite. Ce choix permet de bénéficier de l'accélération matérielle offerte par le GPU du smartphone lorsque celle-ci est disponible, tout en garantissant un comportement stable du modèle lors de l'exécution.

Une quantification du modèle en **INT8** a néanmoins été envisagée. Cette technique permet en théorie de réduire la taille du modèle et d'améliorer les performances sur certains appareils. Cependant, lors de nos essais, cette approche a montré plusieurs limites. La conversion en INT8 entraîne une **forte sollicitation des ressources** lors de la phase de conversion, notamment au niveau du GPU et de la mémoire vive. Elle nécessite également un **ré-entraînement partiel du modèle à partir d'un échantillon du dataset**, ce qui peut provoquer une baisse de précision si l'échantillon utilisé n'est pas suffisamment représentatif.

De plus, l'utilisation d'un modèle quantifié en INT8 aurait impliqué de répéter cette conversion après chaque phase de ré-entraînement. Étant donné les contraintes observées lors de cette étape et les risques en termes de stabilité et de reproductibilité du processus, ce choix s'est révélé peu adapté au cadre du projet.

Ainsi, afin de garantir la fiabilité des détections et de conserver un processus d'entraînement et d'intégration stable, nous avons fait le choix de privilégier un modèle au format **float32**, offrant un meilleur compromis entre performances, précision et robustesse.

### 5.3 Affichage des résultats et expérience utilisateur

L'exécution locale du modèle présente des avantages importants en termes d'expérience utilisateur. L'absence de communication réseau permet d'obtenir des résultats rapides et réguliers, sans dépendre de la qualité de la connexion Internet. Dans le cadre de la détection en temps réel, une latence volontaire d'environ **300 millisecondes** a été introduite entre deux analyses successives afin de limiter la charge de calcul imposée au smartphone et d'assurer la stabilité de l'application. Malgré cette régulation, les objets détectés apparaissent rapidement à l'écran et l'utilisateur ne perçoit pas de temps d'attente gênant lors de l'utilisation.

Un soin particulier a été apporté à l'affichage des résultats de la détection. Les coordonnées des boîtes englobantes fournies par le modèle doivent être adaptées aux dimensions et à l'orientation de l'image affichée dans l'interface Flutter. Des ajustements ont été réalisés afin de corriger les éventuels décalages liés à la rotation de la caméra ou aux changements d'orientation de l'appareil. L'objectif était de garantir un positionnement précis et cohérent des cadres autour des objets détectés.

De plus, l’affichage des informations associées à chaque détection, telles que le nom de l’objet et le score de confiance, a été conçu de manière à rester lisible en toutes circonstances. Par exemple, lorsque l’objet détecté se situe près du bord supérieur de l’écran, le texte est automatiquement repositionné afin d’éviter qu’il ne sorte de la zone visible.

Ainsi, l’intégration du modèle TensorFlow Lite dans l’application Flutter ne s’est pas limitée à un simple aspect technique. Elle a nécessité un ensemble de choix et d’optimisations visant à concilier performances, précision et confort d’utilisation. Ces efforts ont permis de proposer une application capable de détecter des objets scolaires en temps réel, de manière fluide, autonome et adaptée aux contraintes matérielles des smartphones.

# PARTIE 6 – ANALYSE CRITIQUE, DIFFICULTÉS RENCONTRÉES ET PERSPECTIVES

## 6.1 Démarche itérative et difficultés techniques rencontrées

Tout au long de cette SAE, le projet a avancé de manière progressive. Les choix techniques ont été ajustés au fur et à mesure de l'avancement, en fonction des résultats obtenus et des contraintes rencontrées. Cette approche nous a permis d'améliorer progressivement la solution et de rester cohérents avec les objectifs du projet.

La principale difficulté rencontrée a concerné la constitution du dataset. La collecte des images, leur tri et leur préparation ont demandé beaucoup plus de temps que prévu. Il a fallu s'assurer que les images soient suffisamment variées, bien annotées et représentatives des situations réelles d'utilisation. Cette étape a été longue, mais elle était indispensable pour obtenir un modèle capable de distinguer correctement des objets scolaires parfois très proches visuellement.

Une autre contrainte importante a concerné le ré-entraînement du modèle. Le projet repose sur la détection d'objets de petite taille, ce qui nécessite l'utilisation d'images de résolution relativement élevée. Ce choix améliore la précision, mais rend le ré-entraînement plus long et plus coûteux en ressources. Contrairement à certains projets utilisant des images de plus faible résolution, le ré-entraînement ne pouvait donc pas être effectué rapidement ou automatiquement.

Enfin, tout au long du projet, un compromis a dû être trouvé entre précision et performances. Améliorer la précision du modèle impliquait des choix techniques plus exigeants, qui avaient un impact sur le temps d'entraînement et sur les contraintes liées à l'exécution sur mobile. Ces choix ont été assumés, car l'objectif principal du projet était de proposer une application fiable et utilisable, plutôt qu'une solution rapide mais peu précise.

En résumé, les principales difficultés rencontrées ont été liées aux données, à l'entraînement du modèle et aux compromis techniques nécessaires pour adapter une solution d'intelligence artificielle à une application mobile.

## 6.2 Compromis techniques et limites du projet

Malgré le bon fonctionnement de l'application, certaines limites existent. Ces limites ne concernent pas les fonctionnalités principales, mais plutôt des aspects qui auraient demandé plus de temps ou une mise en place plus complexe.

La principale limite du projet concerne le ré-entraînement du modèle d'intelligence artificielle. Celui-ci ne peut pas être effectué directement depuis l'application mobile. Le ré-entraînement doit être réalisé dans un environnement externe, avec des outils et des ressources adaptés. Cette contrainte est liée à la complexité du processus et à la taille des images utilisées pour la détection des objets scolaires.

Dans notre cas, la détection d'objets de petite taille nécessite des images de bonne résolution, ce qui rend le ré-entraînement plus long et plus coûteux en ressources. Intégrer cette étape directement dans l'application aurait compliqué le projet et aurait pu avoir un impact négatif sur la précision du modèle.

Ce compromis a été assumé afin de privilégier une application stable, fiable et fonctionnelle, plutôt que d'ajouter des fonctionnalités avancées difficiles à maîtriser dans le temps imparti. La structure du projet reste néanmoins suffisamment claire pour permettre des évolutions futures si nécessaire.

### 6.3 Perspectives d'amélioration et conclusion globale

Les perspectives d'amélioration du projet sont nombreuses. Une première piste consisterait à enrichir davantage le dataset avec des images prises dans des conditions réelles d'utilisation, afin d'améliorer la robustesse du modèle face à des environnements variés et à des arrière-plans complexes. Cela permettrait notamment de réduire encore le nombre de faux positifs et d'améliorer la généralisation du modèle.

Des optimisations supplémentaires pourraient également être envisagées au niveau du modèle, par exemple à travers une quantification maîtrisée ou l'utilisation de délégations matérielles spécifiques, afin de réduire la consommation énergétique tout en conservant une précision satisfaisante. Ces améliorations seraient particulièrement intéressantes dans le cadre d'une utilisation prolongée de l'application sur smartphone.

Enfin, une évolution possible du projet serait la mise en place d'un mécanisme permettant d'exploiter les images partagées par les utilisateurs pour améliorer progressivement le modèle, sous forme de ré-entraînement contrôlé. Cette approche ouvrirait la voie à une application plus évolutive, capable de s'adapter aux usages réels et aux besoins spécifiques des utilisateurs.

## PARTIE 7 – CONCLUSION

Cette SAE nous a permis de réaliser un projet complet mêlant développement mobile et intelligence artificielle, en partant d'une idée initiale jusqu'à la conception d'une application fonctionnelle. Le projet ne s'est pas limité à l'implémentation technique, mais a nécessité une réflexion globale sur les choix technologiques, les contraintes matérielles et l'expérience utilisateur.

La mise en place du modèle de détection d'objets nous a permis de mieux comprendre l'importance des données dans un projet d'intelligence artificielle. La constitution du dataset, son nettoyage et son organisation ont représenté une part importante du travail, mais ont été déterminants pour obtenir des résultats fiables. Ce travail a montré que la qualité des données a un impact direct sur les performances du modèle, en particulier lorsque les objets à détecter sont visuellement proches.

L'intégration du modèle dans une application mobile nous a également confrontés aux contraintes réelles des smartphones. Il a fallu trouver un compromis entre précision, performances et stabilité, notamment en ce qui concerne la résolution des images, la gestion de la latence et le format du modèle utilisé. Ces choix ont été faits dans l'objectif de proposer une application fluide, utilisable et adaptée à un usage réel, plutôt qu'une simple démonstration technique.

Au-delà des aspects techniques, ce projet nous a permis de développer une démarche plus méthodique et réaliste du développement informatique. Nous avons appris à analyser des contraintes, à faire des choix techniques justifiés et à accepter certains compromis en fonction du contexte et du temps imparti. Cette SAE constitue ainsi une expérience formatrice, en lien direct avec les compétences attendues en BUT Informatique, aussi bien sur le plan technique que méthodologique.