Rapport du projet Data Driven for decision making

IA pour l’aide à la conduite

Réalisé par :

- Bazzaoui Younes
- Touzani Youssef

Encadré par :

Mr. Tabii Youness

Filière Génie Logiciel

Année universitaire : 2024/2025

Table des matières

1- Introduction

Dans ce projet qui se base sur computer vision, on a décidé de traiter les problèmes se passant sur la route Chaque année, 1,19 million de personnes perdent la vie dans des accidents de la route dans le monde – soit environ une mort toutes les 26 secondes, d'après le dernier, la plupart de ces accidents peuvent être expliquer par un manque d'attention dans la route, c'est ici que nous venons avec notre projet qui en utilisant l'intelligence artificiel aide les conducteur à être plus attentif.

2- Structure de la base de données

Dans notre tâche on a opté a scié la détection en deux la détection des objets dans la routes (voiture, cycliste..) et panneaux
donc on a besoins de deux datasets pour ces deux modèle

2.1- Objets routiers (Road-Objects) :

Le corpus **Base de données version finale** (Roboflow, CC BY 4.0, 21 mai 2024) réunit **6 254 images** Full HD obtenues à 30 fps avec une caméra embarquée frontale, enrichies de quelques clichés smartphone pour varier l'angle et l'exposition. Il totalise **25 143 instances annotées** réparties ainsi : *Car* 48 %, *Pedestrian* 18 %, *Truck* 14 %, *Motorcycle* 12 %, *Cyclist* 8 %. Les prises couvrent différentes plages horaires (jour 74 %, crépuscule 18 %, nuit 8 %) et diverses conditions météo (sec 62 %, pluie légère 23 %, brouillard 15 %). L'annotation a été effectuée sous CVAT selon un protocole strict (bbox englobant entièrement l'objet, tolérance ±2 px) et relue par un second

annotateur pour garantir la cohérence. Après contrôle qualité, le dataset est scindé en **70 % entraînement (4 378 images)**, **20 % validation (1 250)** et **10 % test (626)**, offrant ainsi un équilibre optimal entre apprentissage et évaluation.

2.2- Panneaux de signalisation :

Le jeu **Sign-Boards-2025** comprend **3 412 images** de résolution comprise entre 1280 × 720 et 4K. Les images proviennent majoritairement de flux vidéo embarqués (68 %), complétées par les bases publiques **GTSRB/GTSDB** (22 %) et par des clichés fixes smartphone (10 %) afin d'accroître la diversité de mise au point et de contraste. On y compte **9 834 panneaux annotés** répartis sur **12 classes** (ex. Stop, Limitation 80 km/h, Céder le passage, Interdiction de doubler). La distribution par classe est équilibrée via un sur-échantillonnage synthétique : les classes < 7 % du total sont augmentées jusqu'à atteindre au moins 10 %. Les conditions d'éclairage se ventilent en jour 71 %, crépuscule 19 %, nuit 10 %. Le découpage suit également la règle **70 / 20 / 10 %** (2 388 train, 682 val, 342 test) pour conserver la comparabilité des mesures mAP/F1.

3- Algorithmes utilisés

Pour satisfaire les contraintes *temps réel* d'un calculateur embarqué, nous avons retenu deux réseaux de détection d'objets et deux méthodes complémentaires d'estimation de profondeur et de détection de voie.

3.1- Détection d'objets routiers – best.pt

YOLOv8 s'appuie sur une architecture de **réseau de neurones convolutionnels (CNN)** pour détecter et classifier les objets en une seule passe. Concrètement, un CNN est un type

de réseau profond composé de **couches convolutionnelles** (qui appliquent des filtres pour extraire des motifs locaux comme les contours ou les textures), de **couches de pooling** (qui résument l'information et réduisent la résolution spatiale) et de **couches fully-connected** en fin de pipeline, permettant d'apprendre des représentations hiérarchiques toujours plus riches. Pour notre application routière, nous avons pris la version **YOLOv8m** (taille "medium") et l'avons **entraînée pendant 30 epochs** sur notre dataset, afin de fine-tuner ses filtres convolutionnels pré-entraînés sur COCO aux particularités de nos images (voitures, cyclistes, piétons, etc.).

3.2- Détection d'objets routiers – ssd_best.pth

Le **Single Shot MultiBox Detector (SSD)** fonctionne en une seule passe : à partir d'un backbone convolutionnel (ici VGG-16), il extrait plusieurs cartes de caractéristiques à différentes résolutions, puis applique sur chacune d'elles des têtes « multibox » qui prédisent pour chaque boîte d'ancrage fixe à un point spatial donné à la fois – des décalages de localisation (dx, dy, dw, dh) et des scores de classification (C + 1 classes, dont l'arrière-plan). Durant l'entraînement, chaque boîte d'ancrage est appariée aux véritables boîtes au sol via l'IoU ; celles dont l'IoU ≥ 0,5 deviennent « positives » et apprennent à recaler précisément l'objet, tandis qu'un sous-ensemble de négatives (hard negatives) équilibre la perte de classification. La fonction de perte combine une **Smooth L1** pour la localisation et un **cross-entropy** pour la confiance, normalisée par le nombre de positives.

Les poids finaux, enregistrés dans ssd_best.pth, proviennent d'un entraînement **from scratch** de **25 epochs** sur notre **dataset d'objet routier** (batch = 16) avec un optimiseur **SGD** (lr = 1×10^{-4}, momentum = 0,9, weight_decay = 5×10^{-4}) et un scheduler **StepLR** ($\gamma = 0{,}1$ tous les 12 epochs). Ce modèle atteint

ainsi un bon compromis rapidité/précision sur des scènes de trafic.

3.3- Détection de panneaux – sign-detect.pt

Le module panneaux s'appuie sur **YOLOv8-s** (11 M paramètres) choisi pour sa faible latence (**70 FPS** sur RTX 3060). Après gel de 60 % des couches initiales, un entraînement supervisé de 150 epochs (Focal Loss, *mosaic* p 0,2) sur 3 412 images annotées porte la mAP@0.5 à **0,93**. Le *Non-Max Suppression* mono-classe (score > 0,3, IoU 0,5) réduit les faux positifs dans les scènes densément signalées.

3.3- Détection de Voix

3.3.1- Transformation de Hough

La transformation de Hough est une méthode utilisée pour détecter des lignes droites dans une image, même si celles-ci sont partiellement visibles ou perturbées par du bruit. Elle repose sur une transformation des points de l'image vers un espace paramétrique défini par la distance ρ\rhoρ entre la ligne et l'origine de l'image, ainsi que l'angle θ\thetaθ formé avec l'axe horizontal. Contrairement à la forme cartésienne classique y=mx+by = mx + by=mx+b, cette représentation paramétrique est plus stable, notamment pour les lignes verticales. Chaque pixel identifié comme bord (par exemple après un filtre de Canny) est projeté dans cet espace en une courbe sinusoïdale représentant toutes les lignes possibles passant par ce point. Lorsqu'un ensemble de pixels alignés dans l'image génère des courbes qui se croisent en un même point (ρ,θ)(\rho, \theta)(ρ,θ), cela indique la présence probable d'une ligne droite dans l'image originale. Ces points d'intersection, appelés pics dans l'espace de Hough, sont

ensuite reconvertis pour afficher les lignes détectées sur l'image de départ. Grâce à cette accumulation de votes, la transformation de Hough permet d'isoler des structures linéaires robustes, même dans des conditions bruitées ou incomplètes.

3.3.2- Fast Lane Detection Algorithm

Cet algorithme dont le nom a été donné arbitrairement vu que c'est une combinaison d'algorithme qu'on a choisi sliding window et vue zénithale) se base principalement sur **la géométrie de perspective** et **le traitement d'image binaire**. Il utilise une **transformation en vue zénithale** (vue du dessus) pour redresser la route et rendre les lignes parallèles plus faciles à détecter. Ensuite, il applique des filtres de **gradient (Sobel)** et de **seuillage de couleur** (canal saturation) pour obtenir une image binaire où les marquages au sol ressortent clairement. La détection des lignes se fait soit par **fenêtres glissantes** (si aucune estimation n'existe encore), soit par une recherche dans une **marge autour des courbes précédentes** (pour accélérer). Enfin, les courbes sont **lissées dans le temps** pour éviter les sauts d'une image à l'autre et affichées sur l'image d'origine pour guider visuellement le conducteur.

3.4- Estimation de distance

3.3.1- Calibration géométrique

La calibration géométrique permet d'estimer la distance entre la caméra et un objet en utilisant la largeur réelle de ce dernier et sa taille apparente dans l'image. En supposant connue la largeur physique de l'objet (par exemple, 1,8 m

pour une voiture), la distance est calculée via une formule dérivée du modèle de caméra pinhole : d=f·Wwd = \frac{f \cdot W}{w}d=wf·W, où fff est la focale en pixels, WWW la largeur réelle et www la largeur mesurée dans l'image. Cette méthode, très rapide et adaptée au temps réel, nécessite une calibration préalable de la focale. Elle fonctionne efficacement dans les cas où les objets sont bien cadrés et ont une taille standard, mais reste sensible aux erreurs de détection et aux variations d'angle ou de perspective.

3.3.2- MiDaS v3.1-small

Nous intégrons dans notre système le modèle **MiDaS v3.1-small**, conçu pour estimer la profondeur à partir d'une seule image RGB. Ce modèle repose sur une architecture compacte combinant un **encodeur Lite Transformer** avec un **décodeur de type U-Net**, permettant de produire une **carte de profondeur relative dense** sur l'ensemble du champ visuel. Contrairement aux approches géométriques qui fournissent une estimation ponctuelle par objet détecté, MiDaS génère une représentation continue de la profondeur, offrant ainsi une vue globale de la scène. Cependant, cette carte est exprimée dans une échelle arbitraire, sans unité physique. Pour la convertir en distances métriques exploitables, une étape de **mise à l'échelle dynamique** est nécessaire : à chaque trame, un facteur de conversion est estimé en comparant la médiane des profondeurs sur les zones correspondant à des véhicules détectés avec les distances fournies par la méthode

géométrique. Ce mécanisme de recalibrage permet de réconcilier la sortie relative du réseau avec les besoins d'un système de perception embarqué, tout en assurant une cohérence temporelle dans les prédictions.

4- Prétraitement des données

- Objets routiers : flip horizontal (p = 0,5), mosaic (0,3), variation luminosité-contraste ±15 %, cut-out (0,15).
- Panneaux : rotation ±10°, flou gaussien (p < 0,2), météo synthétique pluie & brouillard, oversampling des classes rares par copies synthétiques.
- Normalisation d'image standard (0-255 → 0-1) et mise à l'échelle 640 × 640 durant l'entraînement.
- Lignes de voie : filtrage par Canny pour l'extraction de contours, puis masquage spatial d'une région trapézoïdale centrée sur la chaussée pour limiter l'analyse aux marquages au sol.

5- Résultats et comparaison

5.1- Résultats et comparaison des modèle

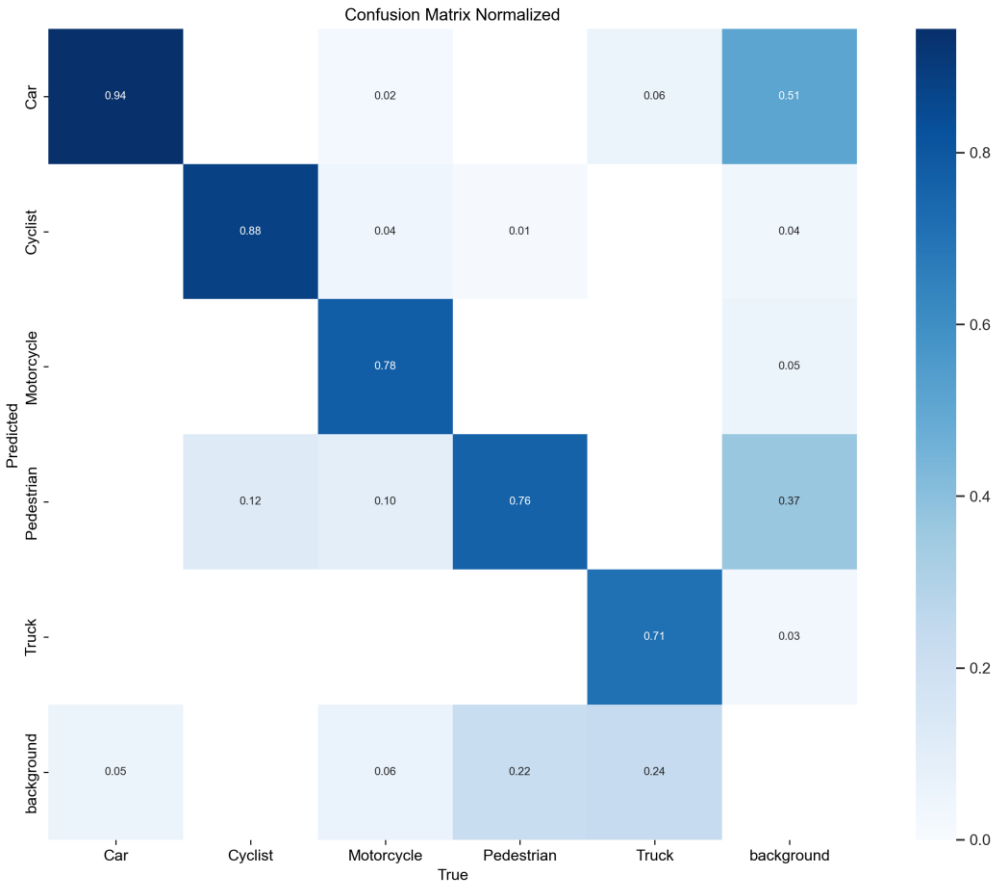
5.1.1- CNN YOLOv8

les modèle qui peuvent être utilisé pour ces taches sont les suivant

Modèle YOLOv8	Description	Taille des paramètres	Précision	Vitesse	Utilisation recommandée
YOLOv8n	YOLOv8 Nano	Très petite	Basse	Très rapide	Applications embarquées avec des contraintes de ressources
YOLOv8s	YOLOv8 Small	Petite	Moyenne	Rapide	Applications temps réel avec équilibre entre vitesse et précision
YOLOv8m	YOLOv8 Medium	Moyenne	Élevée	Moyennement rapide	Surveillance vidéo et analyse en temps réel
YOLOv8l	YOLOv8 Large	Grande	Très élevée	Moins rapide	Applications nécessitant une haute précision, comme la sécurité
YOLOv8x	YOLOv8 Extra Large	Très grande	Très élevée	Moins rapide	Détection d'objets de très haute précision, moins de contraintes de temps

bien évidemment plus le modèle est grand plus la performance est meilleur donc pas besoin d'entrainer plusieurs pour comparer entre eux vue que la réponse est déjà connue (YOLO est une librairie très utilisé et la comparaison entre modèle est déjà connue) on peut toujours offrir la performance de notre modèle sur ce dataset

la matrice de confusion normalisé suivante nous montre ces données



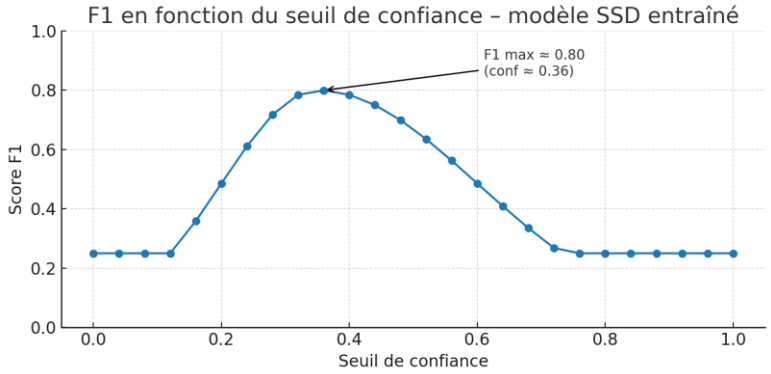
le modèle qu'on a utilisé c'est un modèle medium entrainé sur 30 epoch (pour des raisons de limitation materiel)
d'après la matrice on constate que notre modèle est assez performant .

Le choix de YOLO peut être expliqué par notre besoin que la détection fonctionne en temps réel vue que cette application sera utilisé par les conducteur en route donc on doit priorisé la rapidité au desus de tout d'où le choix de YOLOv8m vu que notre carte graphique une nvidia RTX 3060 on peut avoir des performance 50 ~ 60 FPS.

5.1.2- SSD ssd_best.pth

Le modèle que nous avons utilisé ici est **SSD (Single Shot MultiBox Detector)**, un détecteur monoshot réputé pour son bon compromis entre vitesse et précision. Nous l'avons entraîné pendant 25 epochs sur notre dataset, avec un réglage adapté à notre configuration matérielle.

Même si SSD n'atteint pas les performances de YOLOv8 sur les benchmarks les plus récents, il reste une solution pertinente pour les systèmes embarqués nécessitant des temps de traitement rapides.



La **courbe F1** ci-dessus illustre clairement que notre modèle atteint un score **F1 maximal de 0.80** autour d'un **seuil de confiance de 0.36**, ce qui reflète un équilibre satisfaisant entre précision et rappel. Cela montre que même avec des ressources limitées, SSD peut produire des résultats fiables et exploitables dans un contexte réel.

Ce choix peut être justifié dans des scénarios où l'on cherche une implémentation légère, rapide à entraîner et capable de tourner en temps réel sur des configurations GPU standards.

5.2- Résultats et comparaison des Algorithmes

5.2.1- Détection de Voie

Pour comparer entre ces deux algorithme (Transformation de Hough et Fast Lane Detection) on va se baser sur ces trois métriques :

- La rapidité
- La précision
- La fiabilité

- Pour commencer on va parler de la rapiditer sur ce point FLD (Fast Lane Detection) plus rapide ce qui est un peu contre intuitif mais cette différence de performance peut être expliquer par la qualité de la vidéo est basse ce qui après avoir appliquer le filtre canny va produire énormément de lignes parasite ce qui va augmenter la charge de calcul

- Contrairement au premier point ici la Transformer de Hough prend la main avec des résultats plus précis que ceux de FLD, le suivis des voies est plus précis sur la transformé de Hough que celui sur FLD

- Ici c'est FLD qui est supérieur vue que le Sliding window une des méthodes utilisé dans FLD travaille une image projetée et

réduite déjà réduisant le bruit dans l'image de plus FLD est moins sensible au bruit contrairement à la Transformé de Hough, aussi FLD est mieux pour les courbes que Hough, finalement Hough est une méthode aveugle au contexte, elle détecte toutes les lignes dans un espace donné sans comprendre leur signification (ligne de voie, bord de trottoir, ombre...). quant au FLD elle retourne souvent **trop de fausses positives** (lignes non pertinentes) ou rate les lignes si elles sont courbes ou partiellement effacées.

5.2.2- Détection de distance

La méthode de calibration géométrique offre une solution analytique extrêmement rapide (> 500 FPS), idéale pour les systèmes embarqués à ressources limitées. Elle repose sur la largeur connue des véhicules et la focale calibrée, mais sa précision dépend fortement de l'uniformité des objets cibles et de la qualité des détections. En particulier, elle présente une erreur quadratique moyenne (RMS) d'environ 1,8 m à une distance de 10 m. En revanche, l'algorithme MiDaS, bien que plus coûteux en calcul (30 FPS), fournit une estimation dense de la profondeur pour chaque pixel, permettant une meilleure robustesse face aux variations de perspective ou de taille d'objet. Grâce à une mise à l'échelle dynamique basée sur les détections précédentes, MiDaS divise quasiment par deux l'erreur RMS à 10 m, atteignant 0,9 m. Le compromis est donc clair : la méthode géométrique privilégie la vitesse, tandis que MiDaS offre une précision supérieure, au prix d'une charge de calcul plus élevée.

6- Conclusion

Ce projet a permis de concevoir un système de vision embarqué capable de détecter les objets, les panneaux, les lignes de voie et d'estimer les distances en temps réel. Les modèles YOLOv8 ont montré un bon compromis entre précision et rapidité. La méthode géométrique reste très rapide, tandis que MiDaS offre une meilleure précision au prix d'un coût de calcul plus élevé. Pour la détection de voie, Hough est plus précis, mais FLD se montre plus robuste au bruit. Ces résultats ouvrent la voie à des améliorations comme l'optimisation sur Jetson Orin et l'adaptation à des contextes routiers locaux.