

Rapport de SY32 - P2024

Martin C., Sacha S.

02/10/2024

Introduction

La détection et la reconnaissance de panneaux de signalisation jouent un rôle crucial dans le développement de systèmes avancés d'assistance à la conduite (ADAS) et de véhicules autonomes. La capacité à identifier avec précision les panneaux de signalisation permet non seulement de renforcer la sécurité routière, mais aussi d'améliorer l'efficacité des véhicules intelligents.

Ce projet vise à créer une base de données d'images de panneaux de signalisation et à concevoir un système de reconnaissance performant. Il a été réalisé en binôme et est composé de deux phases, la première avec un accent particulier sur l'utilisation de méthodes de machine learning classiques pour détecter puis classifier, la seconde se concentre sur l'application de techniques d'apprentissage profond via des réseaux convolutifs pour de la détection, de la segmentation et de la classification.

Le projet s'inscrit dans le cadre du Challenge étudiant UTAC, la première compétition européenne en environnement réel dédiée à la mobilité future, autonome et connectée. Ce challenge, soutenu par la Société des Ingénieurs de l'Automobile (SIA), permet aux équipes universitaires du monde entier de concevoir et de démontrer des véhicules ou des systèmes liés à la mobilité de demain. Les résultats de ce projet apporteront une contribution significative à l'équipe de l'Université de Technologie de Compiègne (UTC) participant à ce défi.

Les sections suivantes présenteront nos choix d'outils méthodologiques, nos différentes approches pour la détection utilisant le machine learning, et enfin les améliorations apportées par l'utilisation des réseaux de neurones convolutifs.

nous avons utilisé GitHub pour la gestion de notre code source et le suivi des tâches. Les fonctionnalités de GitHub, telles que les issues et les milestones, nous ont permis de diviser notre projet en étapes clairement définies et de suivre l'évolution de chaque tâche. Les milestones nous ont aidés à planifier et à visualiser les différentes phases du projet, assurant ainsi une progression structurée et mesurable.

En parallèle, nous avons utilisé Discord comme principal outil de communication. Cette plateforme nous a permis d'échanger facilement des liens d'articles, des morceaux de code, et de partager des ressources en temps réel. De plus, les capacités de Discord en matière de réunions vocales et de visioconférences ont facilité nos discussions, permettant des réunions régulières pour faire le point sur l'avancement du projet, discuter des défis rencontrés, et planifier les prochaines étapes.

1.2 Recherches initiales

Avant de commencer à coder, nous avons choisi de faire un état de l'art des différentes méthodes éprouvées.

N'ayant pas de connaissances préalables sur les méthodes de détection d'objets, nous avons d'abord utilisé une approche logique. La première méthode que nous avons découverte est inspirée de l'article [5], qui segmente l'image en deux couleurs, rouge et bleu, détecte les bords, puis cherche les formes d'ellipses (correspondant à des cercles) et de triangles. Nous avons adapté cette méthode en utilisant la détection de bords basée sur la fonction Canny de la bibliothèque OpenCV, en segmentant l'image en rouge, vert, bleu et orange et en cherchant les formes de triangles, octogones et cercles.

Cependant, cette approche a rapidement montré ses limites. Nous avons donc utilisé l'approche de fenêtre glissante vue en cours. Bien que cette seconde méthode ait donné des résultats prometteurs, elle comportait beaucoup de faux positifs.

Nous avons alors opté pour une troisième méthode, qui utilise la segmentation de couleurs pour limiter les zones à tester. Nous appelons ces masques les régions

1 Contexte de travail

1.1 Méthodologie de travail

Pour assurer un suivi rigoureux et organisé de l'avancement de notre projet, nous avons adopté plusieurs outils de gestion et de communication. Tout d'abord,

d'intérêt (ROI), que nous envoyons ensuite à un classifieur binaire, puis à un classifieur multi-classe. Cette approche s'est révélée la plus prometteuse pour les méthodes classiques de machine learning.

Concernant les approches par réseaux de neurones convolutifs (CNN), nous avons fait un état de l'art des modèles existants que nous avons ensuite revus en cours de SY32. Ces premiers pas vers le Deep Learning nous ont dirigé vers le choix de YOLO (You Only Look Once) en raison de sa popularité et de ses performances élevées en matière de détection d'objets en temps réel. En effet, depuis sa conception, YOLO a été utilisé pour détecter et reconnaître les panneaux de signalisation, les piétons, les feux de circulation, les véhicules, etc. [4]. Nous nous sommes donc dit qu'il conviendrait parfaitement à notre tâche. Cependant, les résultats initiaux de YOLO sur notre jeu de données n'étaient pas satisfaisants, principalement en raison du manque de diversité et de taille du jeu de données d'entraînement et compte tenu de nos limitations en terme de puissance de calcul.

Pour améliorer les performances, nous avons essayé de fine-tuner YOLO en utilisant un ensemble de données plus représentatif et en ajustant les hyperparamètres du modèle. Cette étape a amélioré les résultats. Mais finalement, l'idée du projet était de créer un réseau totalement par nous-mêmes sans utiliser de modèle pré-entraîné.

Nous avons alors exploré une autre approche combinant U-Net et un classifieur. U-Net est un réseau de neurones convolutifs conçu pour des tâches de segmentation d'image, qui génère un masque de segmentation indiquant les régions d'intérêt (ROI) dans l'image. En utilisant U-Net, nous avons pu segmenter précisément les objets d'intérêt dans l'image. Une fois ces régions segmentées, nous avons utilisé un classifieur pour identifier les classes spécifiques des objets détectés. Cette combinaison s'est avérée efficace car U-Net excelle dans la localisation précise des objets, tandis que le classifieur permet une classification fine des objets détectés. Cette approche a permis de réduire considérablement les faux positifs et d'améliorer la précision globale du système de détection.

2 Détection de panneaux par méthodes classiques

Cette section décrit les diverses recherches, tentatives et méthodes employées pour développer un modèle de détection de panneaux de signalisation à l'aide de techniques classiques de machine learning. Nous examinons les approches de segmentation colorimétrique et la re-

cherche de formes géométriques puis l'utilisation de la méthode de la fenêtre glissante combinée avec des descripteurs de forme tels que HOG (Histogram of Oriented Gradients) et enfin une approche par régions d'intérêt

Pour les deux dernières approches, nous utilisons un double classifieur : le premier est binaire (panneau, feu ou rien), et le second est multi-classes (quel type de panneau ou feu).

Nous avons décidé de redimensionner tous les panneaux étiquetés et les fenêtres à prédire au format 32×32 afin d'éviter le fléau de la dimensionnalité. En d'autres termes, nous voulons nous assurer que le nombre de variables prédictives ne soit pas disproportionné par rapport au nombre de données d'entraînement. La fonction de redimensionnement d'OpenCV est ajustée en fonction de la taille de l'image. En cas d'augmentation de la taille (upscale), nous utilisons INTER_LANCZOS4, et en cas de réduction de la taille (downscale), nous utilisons INTER_AREA.

L'analyse des données a révélé un déséquilibre de classes qu'il sera nécessaire de prendre en compte dans nos classificateurs. Comme illustré dans la Figure 1, il y a une sur-représentation des panneaux d'interdiction et une sous-représentation des feux.

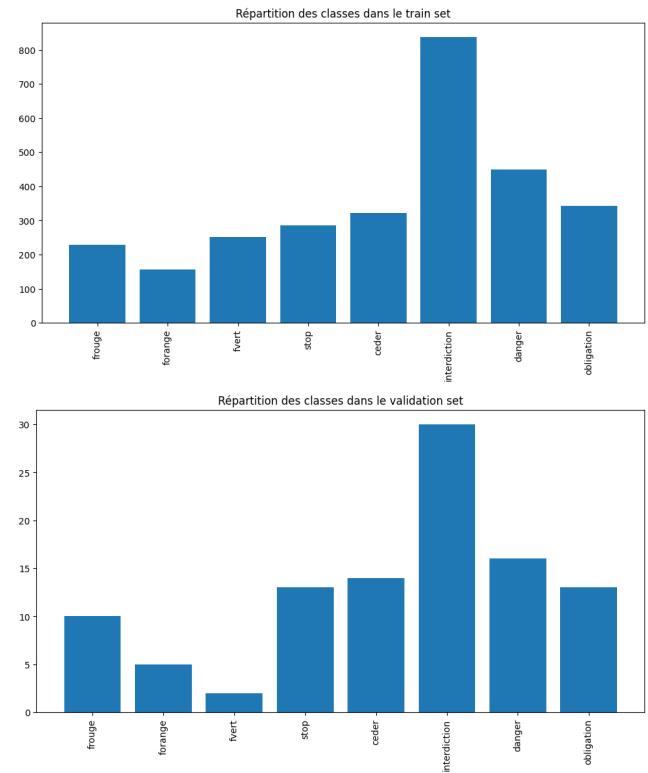


FIGURE 1 – Répartition des données dans le jeu de données d'entraînement et de validation

2.1 Approche : détection de formes

Cette première approche nous est apparue de manière assez évidente dès le début de notre étude de détection. L'idée de base, relativement intuitive, consiste à regrouper les panneaux en se basant sur leur couleur et leur forme. Nous avons cherché à étayer cette méthode en nous appuyant sur des articles de la littérature scientifique, et nous avons trouvé un article pertinent intitulé "Extraction de panneaux de signalisation routière dans des images couleurs" par Soheilian et al. (2010) [[5]].

Dans cet article, les auteurs décrivent un classifieur manuel qui segmente les images par couleur avant de détecter les bords à l'aide de l'opérateur de Canny. Ensuite, ils recherchent des ellipses et des triangles dans les segments colorimétriques. Cette méthode s'est révélée efficace pour la détection de panneaux circulaires et triangulaires. Cependant, nous avons rencontré des difficultés significatives dans la détection des panneaux "Stop" octogonaux. Les variations de luminosité et de perspective ont rendu la détection des formes problématique, la plupart des détections étant interprétées comme des cercles en raison du nombre de côtés d'un octogone.

De plus, la détection de ces formes basée sur l'application d'un détecteur de contours, comme l'opérateur de Canny d'OpenCV, est très sensible au bruit. Nous pouvons réduire ce bruit en appliquant un flou gaussien, mais l'efficacité de cette opération dépend de la qualité de l'image en entrée. En effet, si l'image est déjà légèrement floue, l'application supplémentaire d'un flou gaussien peut rendre la détection des bords presque impossible.

Cette sensibilité au bruit et à la qualité de l'image d'entrée a rendu cette méthode peu fiable pour une application généralisée de détection de panneaux. La Figure 2 compare les résultats de l'opérateur de Canny sans et avec l'application d'un flou gaussien. Sans l'application du flou gaussien (à gauche), l'opérateur de Canny détecte de nombreux contours parasites dus au bruit présent dans l'image. En revanche, avec l'application du flou gaussien (à droite), le bruit est réduit, ce qui améliore la détection des bords. Cependant, comme on peut le remarquer sur la Figure 2, même avec l'application du flou, seuls les panneaux sont détectés, excluant d'autres éléments comme les feux de signalisation, ce qui rend ce détecteur peu fiable.



FIGURE 2 – Détection de bords avec l'opérateur de Canny. À gauche : sans application de flou gaussien, à droite : avec application de flou gaussien.

Nous avons ensuite eu un cours sur une autre approche plus prometteuse, la technique de la fenêtre glissante, que nous allons aborder. Cette méthode permet de balayer systématiquement l'image avec une fenêtre de taille fixe afin de détecter les objets d'intérêt, ce qui offre une plus grande flexibilité et précision dans des conditions variées.

2.2 Approche : fenêtre glissante

L'approche de la fenêtre glissante est une technique de détection d'objets dans laquelle une petite fenêtre de taille fixe est glissée sur toute l'image afin de détecter des objets d'intérêt. Cette méthode nous permet de balayer systématiquement l'image et de capturer les objets, indépendamment de leur position.

Nous avons utilisé l'algorithme HOG (Histogram of Oriented Gradients) pour extraire les caractéristiques de formes de chaque boîte envoyée au classifieur. HOG fonctionne en comptabilisant les occurrences de gradients orientés dans une portion localisée de l'image, ce qui aide à distinguer les formes et les contours des objets.

En utilisant la technique des fenêtres glissantes, nous avons balayé les images avec des fenêtres de différentes tailles : 64×64 pixels pour les panneaux et 32×64 pixels pour les feux de signalisation, avec un pas de 32 pixels. Nous avons ensuite réduit la taille globale de l'image pour générer des boîtes englobantes de tailles plus grandes : 128×128 , 256×256 , 512×512 , et enfin 1024×1024 pixels. Nous avons appliqué la même logique pour les feux de signalisation.

Ce faible pas de 32 pixels a été choisi pour garantir

que nous ne manquions aucun objet d'intérêt, malgré la nécessité de parcourir l'image entière en raison de la diversité des images. En effet, les images prises par les étudiants ne correspondent pas uniquement à des dashcams, elles présentent donc des angles et des environnements différents.

Pour le choix des classifieurs, nous avons utilisé un jeu de données d'entraînement comprenant des images négatives (sans panneaux). Ce jeu de données a été subdivisé en ensembles d'entraînement (80%) et de validation (20%). Les métriques choisies étaient l'accuracy pour évaluer la précision globale, le rappel pour l'exactitude dans la classification des bonnes catégories, et le temps d'exécution. Nous obtenons les résultats suivants :

Modèle	accuracy	precision	fit_time
DecisionTree	0.6111	0.6291	11.3721
KNeighbors	0.6984	0.7286	0.0000
RandomForest	0.8968	0.9224	9.6630
SVM	0.8730	0.8970	55.3947

TABLE 1 – Résultats des différents modèles

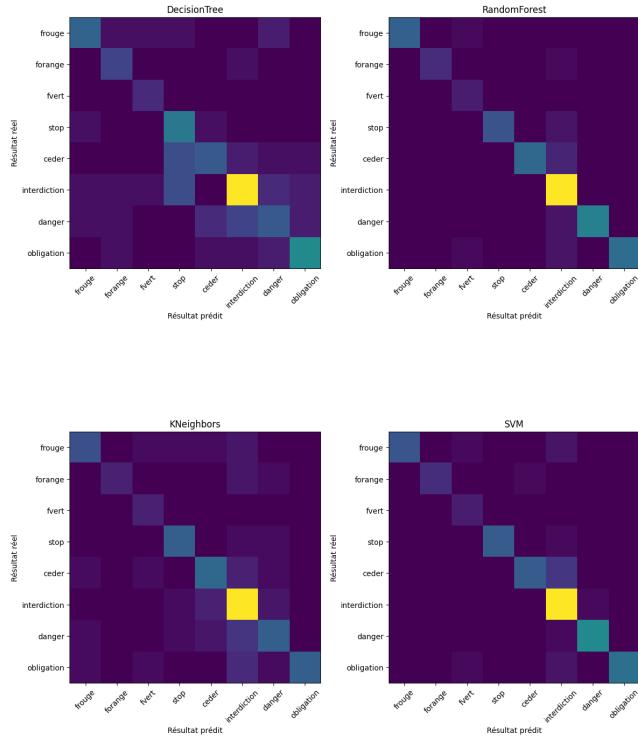


FIGURE 3 – Heatmap des erreurs de classification

Les résultats montrent que les classifieurs SVM et RandomForest sont les plus performants, même si nous pouvons remarquer d'après la heatmap une légère sur-

estimation des panneaux obligation. Nous avons préféré le RandomForest en raison de temps d'entraînement nettement inférieur pour des résultats équivalents. Pour gérer le déséquilibre des classes, nous avons utilisé l'hyper-paramètre `class_weight = "balanced"`.

Nous obtenons les détections suivantes pour une image du jeu de données de validation :

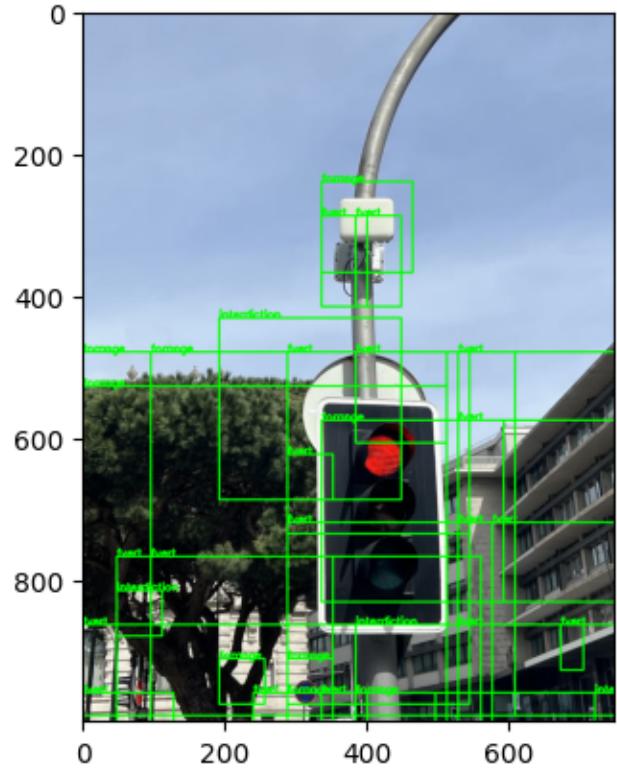


FIGURE 4 – Exemple de détection avec la méthode des fenêtres glissantes.

Nous avons remarqué que les images d'entraînement ont des bounding boxes très serrées autour des panneaux de signalisation. Cela a entraîné des faux négatifs, car la méthode de la fenêtre glissante utilise des boxes parfois beaucoup plus grandes que les panneaux. Pour remédier à ce problème, nous avons choisi d'augmenter la taille des bounding boxes d'entraînement afin qu'elles soient des multiples de 64 × 64 pixels.

En résumé notre méthode par sliding windows suit les étapes :

1. Génération des boîtes à prédire de formes carré et rectangulaire, extraction du HOG de chacune de ces images
2. Prédiction binaire à partir du HOG
3. Prédiction multi-classe à partir du HOG +

Histogramme rouge, vert et bleu

Malgré ses avantages, la méthode de la fenêtre glissante présente plusieurs limitations. Parmi celles-ci, on note un nombre important de faux positifs : par exemple, des branches d'arbres peuvent être détectées comme des triangles, et des pneus peuvent être confondus avec des panneaux circulaires. De plus, le temps d'exécution est élevé, atteignant jusqu'à une minute par image, ce qui rend les prédictions en temps réel peu fiables.

Nous obtenons sur les données de test avec cette méthode les résultats suivant, confirmant le nombre très important de faux positifs :

- AP (sans classification) : 14.05 %
- AR (sans classification) : 08.65 %
- mAP (avec classification) : 16.18 %
- mAR (avec classification) : 09.51 %

Pour réduire ces erreurs et améliorer la performance globale, nous avons adopté une approche par région d'intérêt (ROI). Cette technique consiste à limiter la détection des boîtes à certaines zones spécifiques de l'image où les objets d'intérêt sont plus susceptibles de se trouver. En restreignant ainsi le champ de détection, nous parvenons à réduire le nombre de faux positifs et à accélérer le temps de traitement.

2.3 Approche : région d'intérêt

Dans cette section, nous avons opté pour une approche basée sur la segmentation de couleurs, en convertissant l'image RGB en espace HSV. Cette conversion facilite la segmentation car l'espace HSV permet une meilleure distinction des couleurs, essentielle pour identifier les régions d'intérêt. En nous appuyant sur les résultats remarquables de Fleyeh (2008) et Fleyeh (2005) [[2]; [3]], nous avons optimisé la segmentation des couleurs et les opérations morphologiques appliquées sur les masques, même dans les cas les plus complexes comme du brouillard.

Nous avons subdivisé notre ensemble d'entraînement en deux parties pour évaluer notre méthode. Au cours de cette évaluation, nous avons identifié deux cas où notre méthode n'était pas efficace :

- Images à teinte bleutée (par exemple, ciel gris) : La détection des panneaux rouges était insuffisante.
- Photographies prises de nuit : Les zones détectées étaient trop étendues en raison de l'importance accrue des couleurs émises par les feux la nuit.



FIGURE 5 – Comparaison de bonnes et mauvaises détections avant correction

Pour résoudre ces problèmes, nous avons adapté la fonction de segmentation des images. Le rouge devenant vers du violet clair ou foncé en ces conditions, nous l'avons ajouté lorsque l'image est bleutée. Après segmentation, nous avons extrait les zones d'intérêt en utilisant la fonction bounding rect d'OpenCV, permettant ainsi de soumettre ces zones à un classifieur binaire. Les zones positives ont ensuite été transmises à un classifieur multi-classe. Les deux classificateurs utilisent des forêts aléatoires, avec des hyperparamètres optimisés pour équilibrer les classes.

Nous avons également enrichi notre jeu de données en intégrant des exemples d'images externes, notamment des panneaux provenant du jeu de données référencé [1]. Ces images ont été redimensionnées au format 64 × 64. Cette augmentation nous a permis de mieux gérer les effets de perspective présents sur les panneaux, ces caractéristiques étant mieux traitées par les réseaux de neurones convolutifs.

Les résultats d'entraînement sur les données de test avec un redimensionnement de toutes les images en taille 32 × 32 pour éviter le fléau de la dimensionnalité sont résumés comme suit :

- AP (sans classification) : 72.77%
- AR (sans classification) : 22.73%
- mAP (avec classification) : 44.93%
- mAR (avec classification) : 23.57%

En raison d'un nombre élevé de faux positifs, nous avons décidé de ne conserver que les résultats dépassant un seuil spécifique, par exemple pour un seuil de 0.5, ce qui a entraîné une amélioration significative des performances :

- AP (sans classification) : 94.09%
- AR (sans classification) : 17.15%

- mAP (avec classification) : 60.25%
- mAR (avec classification) : 16.84%

Nous obtenons un bon compromis précision / rappel, pour l'enchaînement d'étape suivant :

1. Segmentation des couleurs adapté automatiquement à l'image
2. Application de l'opération morphologique de fermeture sur les masques
3. Génération des boîtes englobantes des différents blobs de chaque masque de taille 32×32
4. Prédiction binaire de ces boîtes à partir de leur HOG
5. Prédiction multi-classes sur les boîtes détectées comme étant des panneaux à partir de la combinaison de leur HOG et de leurs trois histogrammes de couleurs
6. Application du filtre NMS avec un seuil IOU de 0.1 et un score minimum de 0.2

3 Détection de panneaux par méthodes d'apprentissage profond

Cette partie développe les travaux réalisés afin d'implémenter notre outil de détection de panneaux de signalisation à l'aide d'algorithmes d'apprentissage profond. Elle développera les différentes approches réalisées, les différents modèles employés, leurs architectures et quelques visualisations de leurs résultats.

3.1 Introduction

Dans le domaine de la vision par ordinateur, plusieurs modèles de deep learning ont été développés pour résoudre des tâches complexes telles que la classification d'images, la segmentation d'images et la détection d'objets. Nous avons eu l'occasion d'étudier ces différents modèles en cours de SY32. L'implémentation devant être réalisée complètement par nous-mêmes. L'idée était alors de reproduire l'architecture de l'un des réseaux étudiés et de l'adapter à notre cas d'étude.

Étant donné que le critère de rapidité d'exécution est un critère important dans le cadre du challenge, nous avons décidé de partir sur une architecture semblable au modèle YOLO qui traite l'ensemble de l'image en une seule passe, ce qui le rend particulièrement rapide par rapport aux méthodes basées sur les régions d'intérêt comme R-CNN.

3.2 Première approche : YOLO

3.2.1 Implémentation

Format YOLO adopte une approche différente des autres modèles de Deep Learning : il ne regarde l'image entière qu'une seule fois, d'où son nom. Pour ce faire, YOLO divise l'image d'entrée en une grille de taille $S \times S$. Pour chaque cellule de la grille, il prédit B cadres de délimitation ($B=3$ signifie qu'une cellule de la grille peut contenir au plus 3 objets). Chaque cadre délimitant l'objet est défini par 4 coordonnées, par un score de confiance, et des probabilités associées aux différentes classes, soit $4 + 1 + C$ composantes. Chaque cellule ne pouvant contenir au plus B objets, celle-ci est alors définie par un vecteur de taille $B \times (5 + C)$.

$$\begin{pmatrix} x_{\text{centre_obj1}} \\ y_{\text{centre_obj1}} \\ long_{\text{obj1}} \\ haut_{\text{obj1}} \\ \vdots \\ x_{\text{centre_objB}} \\ y_{\text{centre_objB}} \\ long_{\text{objB}} \\ haut_{\text{objB}} \\ conf_{\text{obj1}} \\ \vdots \\ conf_{\text{objB}} \\ classe1_{\text{obj1}} \\ \vdots \\ classeC_{\text{obj1}} \\ \vdots \\ classe1_{\text{objB}} \\ \vdots \\ classeC_{\text{objB}} \end{pmatrix}$$

L'image possédant $S \times S$ cellules est alors définie par un vecteur de dimension $(S, S, B \times (5 + C))$. La figure 6 illustre ce découpage. Ici, on a S qui vaut 3, B vaut 1, et C correspond à notre nombre de classes soit 8.

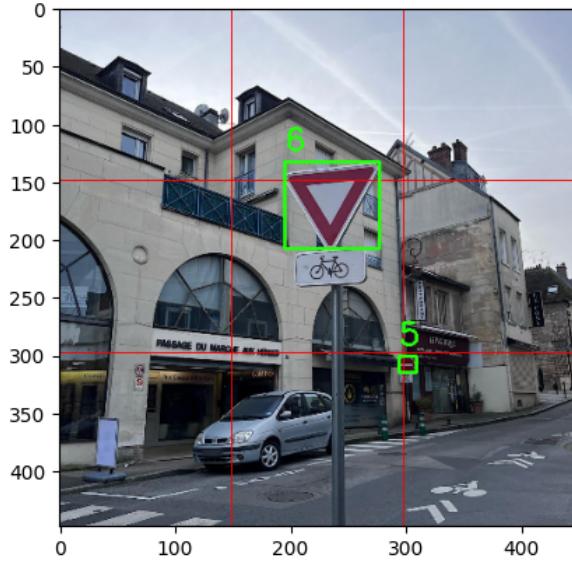


FIGURE 6 – Division de l'image par YOLO

Modification Pour préparer notre dataset, nous avons suivi les étapes suivantes :

- 1. Formatter les annotations** : Il a fallu transformer les annotations des fichiers CSV afin de correspondre au format accepté par notre modèle de YOLO, c'est-à-dire de les organiser selon le format $(S, S, B \times (5 + C))$.
- 2. Augmentation des données** : Pour améliorer la robustesse de notre modèle, nous avons appliqué des techniques d'augmentation des données comme de légères rotations et perspectives, des flips verticaux, des changements de luminosité, et des ajouts de bruit. Cela a permis de générer plus de variations dans les données d'entraînement.

Mise en place Il existe plusieurs versions de YOLO. Nous avons décidé d'implémenter la deuxième version dont l'architecture est illustrée sur la figure 7. 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.

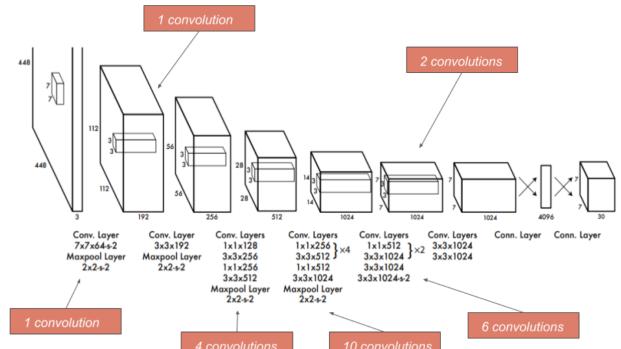


FIGURE 7 – Architecture mise en place

Une fois l'architecture implémenté, il a fallu ajouter sa fonction de perte. Pour rappel, la fonction de perte dit loss function sert à mesurer la différence entre les prédictions faites par le modèle et les valeurs réelles des données. L'objectif principal est alors de chercher à minimiser pendant l'entraînement. Elle est fondamentale pour obtenir de bons résultats car les algorithmes d'optimisation, comme la descente de gradient, utilisent les gradients de la fonction de perte par rapport aux paramètres du modèle pour ajuster ces paramètres de manière à réduire l'erreur. Nous avons donc implémentés la fonction de perte illustrée sur la figure 8.

$$\begin{aligned}
 Loss_{jolo} &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] + \rightarrow \text{Bounding Box coord} \\
 &\quad \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{obj} \left[(C_i - \hat{C}_i)^2 \right] + \lambda_{conf} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{obj} \left[(P_i(C) - \hat{P}_i(C))^2 \right] + \rightarrow \text{Confidence} \\
 &\quad \sum_{i=0}^{S^2} \sum_{j=0}^B \sum_{c \in classes} \left[(P_i(C) - \hat{P}_i(C))^2 \right] \rightarrow \text{Classification}
 \end{aligned}$$

FIGURE 8 – Fonction de perte de YOLO

Cette fonction de perte est divisée en trois partie : celle chargée de trouver les coordonnées du cadre de délimitation, celle chargée de trouver la prédition du score du cadre de délimitation et celle charger de trouver la prédition du score de classe. Tous sont des pertes d'erreur quadratique moyenne et sont modulés par un méta-paramètre scalaire et un score IoU entre la prédition et la vérité terrain.

Cependant, cette implémentation est très demandeuse en puissance de calcul. Afin de pouvoir la faire tourner sur nos machines, il a fallu réduire le nombre d'opérations arithmétiques, notamment en jouant avec la taille des images et les paramètres S et B . Cela a eu un impact sur les résultats, qui sont moins qualitatifs que nous le souhaiterions. Voir figure 9.

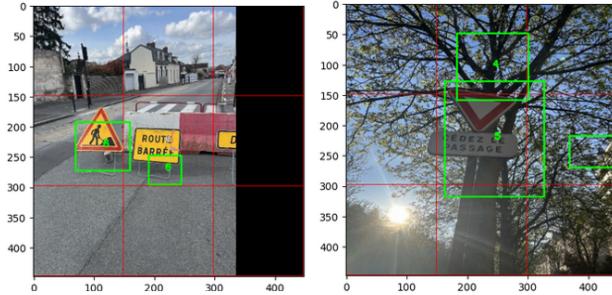


FIGURE 9 – Exemples de détection du modèle

3.2.2 Réglage fin

Après avoir remarqué certaines limitations quant à la puissance de calcul de nos machines, nous avons décidé d'entreprendre un fine-tuning du modèle YOLO, bien moins couteux. En effet, le fine-tuning est une technique d'apprentissage profond utilisée pour adapter un modèle pré-entraîné sur un nouveau jeu de données spécifique à une tâche précise. Cette méthode nous a permis de tirer parti des connaissances déjà acquises par le modèle sur un grand ensemble de données pour améliorer ses performances sur notre tâche de détection de panneaux. Les résultats obtenus sont présentés dans la figure 10.

Les résultats montrent une amélioration notable de la précision des détections, bien que certains défis persistent, notamment en ce qui concerne la détection de petits panneaux ou de panneaux partiellement occultés. Malheureusement, un choix de ne pas autoriser l'utilisation de poids pré-entraînés a été décidé, ce qui a demandé de rechercher une nouvelle approche pour pouvoir détecter nos panneaux.

3.3 Nouvelle approche

3.3.1 Introduction

Face aux limitations rencontrées avec la première approche YOLO, nous avons décidé d'explorer une nouvelle approche basée sur la segmentation d'images. Pourquoi se diriger vers des modèles de segmentation d'objet et non plus de détection ? Etant donné que nos images avaient des panneaux qui pouvaient tout autant prendre toute la place de l'image comme être tout petit dans le fond de l'image. Nous avons décidé d'utiliser un modèle de segmentation afin de mieux gérer ces variations d'échelle, ainsi que les variations de forme des panneaux de signalisation, car ils segmentent directement les pixels pertinents indépendamment de la taille ou de l'orientation de l'objet.



FIGURE 10 – Résultats du fine-tuning

3.3.2 U-Net

Pour la tâche de segmentation, nous avons choisi d'implémenter l'architecture U-Net, qui est bien connue pour ses performances en segmentation d'images médicales et qui peut être adaptée à d'autres domaines. Le modèle U-Net se compose d'une série d'encodages et de décodages symétriques permettant de capturer à la fois des informations globales et locales.

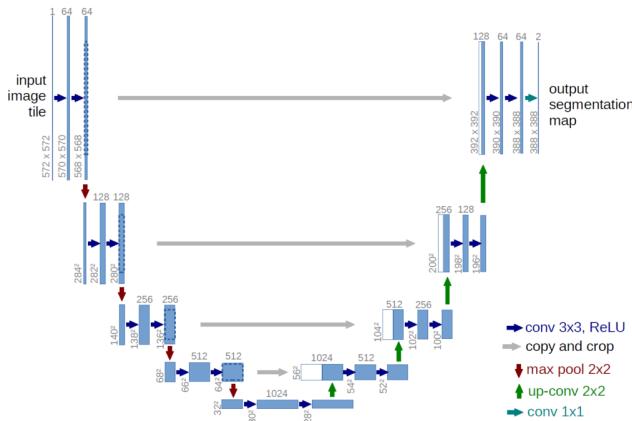


FIGURE 11 – Architecture du modèle U-Net

Le U-Net est illustrée dans la figure 11 et se décompose de la façon suivante :

1. Chemin de contraction (encodeur). Il capture le contexte de l'image en réduisant progressivement la dimension spatiale tout en augmentant la profondeur des caractéristiques. Pour cela, chaque étape de ce chemin se compose de deux convolutions successives avec des filtres de petite taille (3x3), chacune suivie d'une activation ReLU, suivie ensuite d'un max-pooling (2x2) qui réduit la dimension spatiale de moitié tout en doublant le nombre de filtres. Ainsi, à chaque étape de l'encodeur, la résolution de l'image est réduite, mais le nombre de canaux (profondeur des caractéristiques) augmente, permettant au réseau d'extraire des caractéristiques de plus en plus complexes.
2. Chemin d'expansion (décodeur). Il combine les informations contextuelles capturées par l'encodeur avec les détails spatiaux pour produire une segmentation précise. Pour ce faire, chaque étape de ce chemin se compose de convolutions transposées (2x2) pour doubler la dimension spatiale de l'image, de deux convolutions successives avec des filtres de petite taille (3x3), chacune suivie d'une activation ReLU. À chaque étape de l'encodeur, les cartes de caractéristiques de l'étape correspondante du chemin de contraction sont concaténées avec celles du chemin d'expansion (skip connections). Cela permet de récupérer les détails perdus lors des opérations de pooling dans l'encodeur.

3. La dernière couche du modèle U-Net est une convolution 1x1, qui réduit le nombre de canaux de caractéristiques à la profondeur souhaitée pour l'image segmentée. Dans notre cas, nous faisons de la segmentation binaire (panneau ou pas panneau), donc 1 seul canal de sortie. Nous utiliserons ensuite un classifieur pour détecter

Ce modèle prend en entrée nos images et des masques de même taille avec des valeurs binaires, positive quand la coordonnée est présente dans les bounding boxes de nos annotations, 0 sinon.

3.3.3 Classifieur

En complément de la segmentation, nous avons implémenté un classifieur pour attribuer une classe spécifique à chaque région segmentée. En effet, après avoir fourni l'image globale au réseau U-Net, on récupère les zones considérées comme des panneaux/feux. On récupert les boxes de ces zones que l'on va fournir à ce classifieur.

Architecture Ce classifieur commence par une couche d'entrée spécifiant les dimensions des images en entrée. Il enchaîne avec deux paires de couches Conv2D suivies de Dropout pour la régularisation et de Batch-Normalization pour stabiliser l'apprentissage. Chaque couche Conv2D est activée par ReLU pour l'introduction de non-linéarité, suivie de MaxPooling2D pour réduire la dimension spatiale. Les couches Conv2D sont configurées avec 128 filtres dans les deux premières paires et 256 filtres dans les deux paires suivantes. Après les convolutions, les caractéristiques sont aplatis avec Flatten et passées à travers une couche Dense de 512 neurones avec activation ReLU, suivi de Dropout et BatchNormalization pour la régularisation. La dernière couche Dense a 8 neurones avec une activation softmax pour la classification multi-classes.

Les résultats obtenus montrent une bonne capacité de détection et de classification (voir figure 12). Nous regardons maintenant les métriques pour évaluer le modèle.

3.4 Métriques

Cette dernière approche nous a permis d'atteindre un bon niveau de détection. Voici les résultats des différentes métriques :

- AP (sans classification) : 80.40%



FIGURE 12 – Résultats de la classification

- AR (sans classification) : 22.68%
- mAP (avec classification) : 34.97%
- mAR (avec classification) : 40.00%

4 Conclusion

La détection et la reconnaissance de panneaux de signalisation sont des composantes essentielles pour le développement de systèmes avancés d’assistance à la conduite (ADAS) et de véhicules autonomes. Notre projet a exploré deux approches distinctes pour atteindre cet objectif : les méthodes classiques de machine learning et les techniques d’apprentissage profond.

Dans un premier temps, les méthodes classiques ont été mises en œuvre. Nous avons expérimenté diverses techniques telles que la segmentation colorimétrique, la détection de formes géométriques, et la méthode de la fenêtre glissante combinée avec des descripteurs de forme comme HOG. Bien que ces approches aient permis d’obtenir des résultats satisfaisants, elles ont également montré des limitations significatives, notamment en termes de précision et de robustesse face aux variations des conditions d’éclairage et des perspectives.

Par la suite, nous avons exploré les techniques d’apprentissage profond, en particulier l’architecture YOLO (You Only Look Once). Cette approche a démontré une très bonne efficacité en termes de détection et de classification des panneaux de signalisation, grâce à sa capacité à traiter l’image entière en une seule passe. Malgré les défis liés à l’implémentation et aux exigences en puissance de calcul, YOLO a prouvé être une solution prometteuse pour des applications en temps réel, offrant un

bon compromis entre rapidité et précision. Cependant par notre entraînement seul le modèle n’étant pas très performant, contrairement à un transfert learning pour que les poids pré-entraînés répondent à notre problème.

Le projet nous a permis de développer des compétences techniques approfondies dans les domaines de la vision par ordinateur et de l’apprentissage automatique.

En conclusion, bien que les méthodes classiques de détection de panneaux de signalisation offrent une base solide et des résultats acceptables, les techniques d’apprentissage profond, en particulier les réseaux de neurones convolutifs tels que YOLO, représentent l’avenir de la détection d’objets en raison de leur précision et de leur adaptabilité. Le choix d’une segmentation par couleur et l’application d’un fine-tuning de YOLO utilisé dans ce projet contribueront significativement à l’équipe de l’Université de Technologie de Compiègne (UTC) dans le cadre du challenge UTAC.

Références

- [1] A. Caunes. Ftsc : The official french traffic sign classification dataset. <https://github.com/andrewcaunes/FTSC>, 2023.
- [2] H. Fleyeh. Traffic signs color detection and segmentation in poor light conditions. <http://urn.kb.se/resolve?urn=urn:nbn:se:du-1025>, 2005. Conference paper, Refereed, Published.
- [3] H. Fleyeh. Traffic and road sign recognition. <http://urn.kb.se/resolve?urn=urn:nbn:se:du-3396>, 2008. Edinburgh, 2008. URN : urn:nbn:se:du-3396.
- [4] M. Flores-Calero, C. A. Astudillo, D. Guevara, J. Maza, B. S. Lita, B. Defaz, J. S. Ante, D. Zabalal-Blanco, and J. M. Armingol Moreno. Traffic sign detection and recognition using yolo object detection algorithm : A systematic review. <https://www.mdpi.com/2227-7390/12/2/297>, 2024.
- [5] B. Soheilian, A. Arlicot, and N. Paparoditis. Extraction de panneaux de signalisation routière dans des images couleurs. <https://hal.archives-ouvertes.fr/hal-00717032>, 2010. pp. 1-8, France, Jan 2010.