

Projet SY32 — Détection de panneaux de signalisation

Arman Saint-Marc, Ines Abbache

TD1 - Groupe K (qui a corrigé les annotations du dataset) - Mai 2024

Introduction

La conduite autonome, un rêve technologique en passe de devenir réalité, est en train de révolutionner le domaine des transports. Cependant, pour atteindre une véritable autonomie, plusieurs défis restent à surmonter, notamment la détection fiable et précise des panneaux de signalisation.

Dans le cadre de l'UV SY32 : Vision et apprentissage, et du challenge étudiant UTAC auquel participe l'équipe UTC, nous avons exploré et évalué diverses techniques de détection des panneaux de signalisation. Ce rapport vise à présenter les résultats de notre recherche.

Cette étude est structurée en deux parties principales. Dans la première partie, les possibilités offertes par le machine learning. La seconde partie porte sur le deep-learning et l'utilisation de réseaux de neurones convolutionnels.

Il est possible de trouver le projet sur GitLab avec le lien suivant : [GitLabUTC](#).



FIGURE 1 – Exemple de panneaux de tailles différentes

1.2 Différents formats de détection

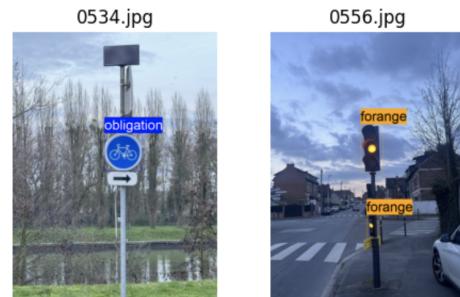


FIGURE 2 – Exemple de deux détections de formats différents

1 Analyse du jeu de données

La visualisation du jeu de données a permis d'identifier les problématiques suivantes :

1.1 Différentes tailles de panneaux de signalisation

Certains panneaux sont situés au premier plan tandis que d'autres se trouvent à l'arrière-plan. Cela entraîne des fenêtres de détection de tailles variables en termes de pixels. Pour standardiser nos tests, il sera nécessaire de reformatter les images afin de les uniformiser. La fonction `resize` de `sklearn` sera utilisée à cet effet.

Dans ce jeu de données, les détections peuvent avoir différentes formes. Certaines sont carrées, d'autres rectangulaires et peuvent être plus allongées verticalement ou horizontalement. Lors de l'entraînement du modèle, il sera nécessaire de prendre en compte ces variations de format. Une solution consiste à ne pas se limiter au contenu de la détection, mais à prendre en compte un carré incluant de la zone détectée. Ce carré sera ensuite redimensionné à une taille standard. Bien que cela puisse entraîner une perte de précision, cette méthode permet une meilleure généralisation lors de l'apprentissage et de la détection.

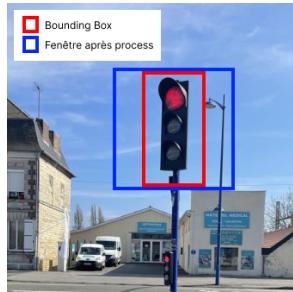


FIGURE 3 – Différence de fenêtre entre détection initiale et après traitement

1.3 Déséquilibre marqué des classes

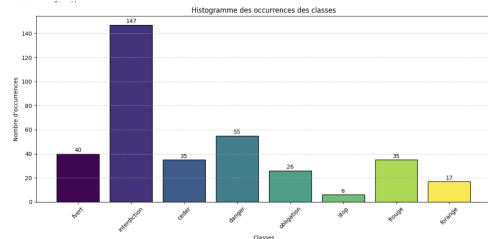


FIGURE 4 – Histogramme des différentes classes

Le jeu de données est déséquilibré, car une image peut contenir plusieurs panneaux, et les panneaux de signalisation d’interdiction apparaissent beaucoup plus fréquemment que les panneaux de stop ou d’obligation. Un jeu de données déséquilibré pose de nombreux problèmes, notamment le risque de sur-apprentissage d’une classe spécifique, ce qui peut fausser le score de précision. En effet, le modèle pourrait apprendre à reconnaître principalement les panneaux d’interdiction, négligeant ainsi les autres classes moins représentées. Cela entraînerait une mauvaise performance du modèle lorsqu’il est confronté à des panneaux de stop ou d’obligation dans des situations réelles. Pour résoudre ce problème, des techniques de rééquilibrage des classes peuvent être utilisées. L’une des méthodes couramment utilisées est le suréchantillonnage des classes minoritaires ou le sous-échantillonnage des classes majoritaires.

1.4 Annotations incorrectes

Lors de la visualisation du jeu de données, une quantité importante de mauvaises annotations a été remarquée. Cela incluait des inversions entre les panneaux d’interdiction et de céder le passage, ainsi que des oubliés de certains panneaux ou feux. Des images ont dû

être mises au format demandé (de 400px à 1000px) et pivotées. Il a été nécessaire de réannoter une grande partie du jeu de données. Sur 800 images, plus de 130 ont dû être réannotées. Plusieurs décisions ont été prises, notamment celle d’annoter tous les panneaux présents sur les images, même les plus petits. Ce choix permet de décider lors de l’apprentissage s’il faut les utiliser ou les ignorer. En effet, il sera possible de supprimer des détections des panneaux en dessous d’une certaine taille en pixels.



FIGURE 5 – Exemple de problèmes d’annotations

2 Détection d’objets avec Machine Learning

Une fois l’analyse approfondie du jeu de données réalisée, il a fallu implémenter la solution de détection de panneaux de signalisation à l’aide du machine learning.

2.1 Enjeux du problème de détection

L’implémentation de la détection de panneaux de signalisation en machine learning nécessite une approche rigoureuse et méthodique.

En plus de l’entraînement et de l’évaluation des algorithmes, il est également important de prendre en compte les contraintes de temps réel et de ressources matérielles lors de l’implémentation. Les algorithmes doivent être suffisamment rapides pour traiter les images en temps réel, et doivent être optimisés pour fonctionner sur des systèmes embarqués tels que les caméras de voiture.

Enfin, il est essentiel de tester et de valider les systèmes de détection dans des conditions réelles, en utilisant des scénarios de test variés et représentatifs. Les tests doivent inclure des situations de conduite normales et anormales, ainsi que des conditions météorologiques et d’éclairage différentes. Normalement, les données de

test fournies sont censées satisfaire ces prérequis de diversité des scènes.

2.2 Traitement du jeu de données

2.2.1 Mise en mémoire des images et labels

La première étape consiste à récupérer et à stocker les images ainsi que les zones de détection associé à leurs classes. Lors de cette première passe, le but est simplement d'insérer les données extraites des fichiers images et CSV dans une liste d'images et une liste de labels.

Cependant, il n'est pas possible d'entraîner un classifieur avec une image entière et une liste de détections sous la forme de $\{(x_{\min}, y_{\min}, x_{\max}, y_{\max}), \text{classe}\}$. Une image contient trop d'informations, souvent inutiles, et une sortie dans ce format est trop complexe. Il est donc nécessaire de résoudre ce problème en découplant les images par zones d'intérêt.

La solution consiste à ne récupérer que les détections des panneaux de signalisation et à les associer à une classe unique. Deux principales décisions ont été prises :

- Ne pas récupérer les détections de moins de 16x16 pixels : ces zones contiennent trop peu de données et sont trop petites sur l'image.
- Récupérer les données d'apprentissage de la même manière que les images inconnues seront analysées à la fin.

La deuxième décision est plus complexe. Dès la réflexion initiale, l'algorithme de fenêtre glissante faisait partie de la solution. Cet algorithme prend une fenêtre de taille définie, ici choisie pour être de dimensions 64x64 pixels. Le but est de récupérer des portions d'image et de vérifier leur classification. En prenant cela en compte, il était judicieux de réutiliser cet algorithme avec les données d'entraînement. Cela permet de résoudre le problème des formats de détection différents et de faire correspondre les données d'apprentissage aux données de prédiction, ce qui est crucial pour obtenir un bon modèle.



FIGURE 6 – Exemple de détection et d'image extraite

Une fonction `getwindow` a été créée pour récupérer le carré correspondant à chaque détection. Par défaut, ce carré est de taille 64x64 pixels, mais il peut être agrandi d'un facteur de 1,5. Ce facteur sera détaillé dans la section sur la détection, mais il correspond à l'algorithme de la pyramide d'images.

Grâce à cela, il est possible d'obtenir une liste d'images de 64x64 pixels chacune associée à une classe.

2.2.2 Augmentation des données

La deuxième étape consiste à augmenter les données d'apprentissage. L'objectif est d'obtenir plus d'images afin d'améliorer l'apprentissage et la généralisation pour chaque classe de notre futur modèle.

Pour ce faire, chaque zone d'intérêt détectée à l'étape précédente est traitée par une fonction qui en retourne plusieurs versions augmentées. Par exemple, l'image redimensionnée à la taille standard de 64x64 pixels, un bruit aléatoire lui est ajouté, elle peut-être inversée horizontalement... Des fonctions changeant la saturation, le contraste, la luminosité sont également utilisées. Les rotations ont été exclues des possibilités en raison des panneaux "cédez le passage" et "danger" qui partagent les mêmes proportions, couleurs et formes, mais diffèrent uniquement par leur orientation.

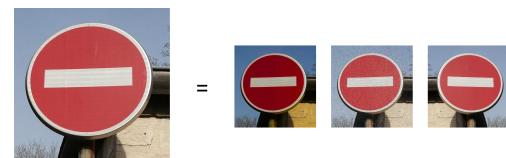


FIGURE 7 – Exemple d'augmentation d'image

2.2.3 Extraction de features

Une image de taille 64x64 pixels reste trop grande pour être utilisée directement dans un modèle de classification. Afin de compresser les informations les plus pertinentes, il a été nécessaire d'utiliser un algorithme d'extraction de caractéristiques. Plusieurs méthodes ont été étudiées pour cette problématique : SIFT, HOG, LBP, GLCM. LBP et GLCM ont très vite été abandonnés car ils étaient respectivement trop peu efficace sur des images aussi complexes ou alors trop gourmands. Une comparaison entre SIFT et HOG a été réalisée :

SIFT (Scale-Invariant Feature Transform) est intéressante pour plusieurs raisons :

- **Détection de points d'intérêt** : SIFT détecte des points d'intérêt dans une image et extrait des

- descripteurs résistants à l'échelle et à la rotation.
- **Résistance aux variations de lumière** : Les descripteurs SIFT sont résistants invariants aux variations d'éclairage.
 - **Limitation** : SIFT est lent.



FIGURE 8 – Exemple d'extraction de caractéristiques avec SIFT

HOG (Histogram of Oriented Gradients) fonctionne de la manière suivante :

- **Division en cellules** : HOG divise l'image en cellules et calcule un histogramme des gradients orientés.
- **Rapidité** : HOG est plus rapide que SIFT.
- **Contours/formes** : HOG est efficace pour capturer les contours et les formes.

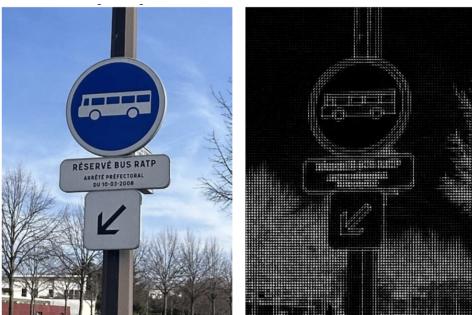


FIGURE 9 – Exemple d'extraction de caractéristiques avec HOG

Après réflexion, HOG est plus intéressant que SIFT car il permet de mieux généraliser les données dans ce cas spécifique. Les contours et formes sont la majeure partie de ce qui différencie un panneau de signalisation avec un autre.

Une combinaison de SIFT et HOG a été utilisée et testée sur le classifieur mais sans résultats convainquants par rapport au temps d'exécution qui a augmenté de manière exponentielle.

	HOG	SIFT	LBP
Accuracy	0.6324	0.2351	0.1047

TABLE 1 – Tableau de comparaison des extractions de caractéristiques

2.2.4 Équilibrage des classes

Une fois les images augmentées et les descripteurs extraits, il était nécessaire d'équilibrer les différentes classes. Comme indiqué précédemment, il y a plus de panneaux d'interdiction comparés aux panneaux stop ou obligation. À chaque extraction de descripteur, le résultat était soit ajouté à la liste d'images augmentées, soit à la liste d'images initiales. Un compteur vérifie quelle classe est la plus représentée et son nombre d'images. Ensuite, pour les autres classes, les images initiales sont utilisées en premier, puis les images augmentées pour combler les lacunes de chaque classe.

En procédant ainsi, les classes sont rééquilibrées comme suit :

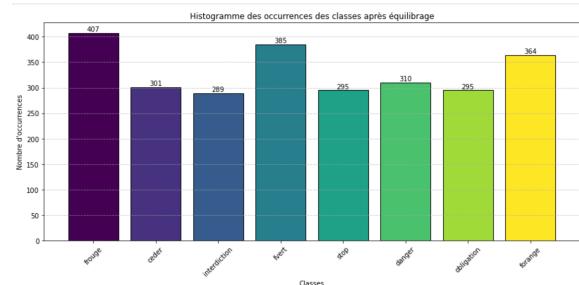


FIGURE 10 – Homogénéisation des proportions des classes

2.2.5 Considération d'une classe vide

Après ces étapes, il était possible de commencer à entraîner un classifieur. Cependant, dans le cadre de la détection de panneaux de signalisation, il est important de considérer un cas où l'image ne comporte aucun panneau. Pour cela, une solution a été trouvée : utiliser les images sans labels et extraire deux "zones d'intérêt" par image. De même, sur les images non vides, une fonction est chargée de prendre une zone de taille variable et au hasard dans l'image en vérifiant qu'elle ne touche aucun panneau. L'objectif est de récupérer des zones aussi variées que possible.

Ces images vides ne sont pas augmentées car elles sont déjà en nombre suffisant et seront traitées comme les autres images. La classe vide, nommée "empty", sera

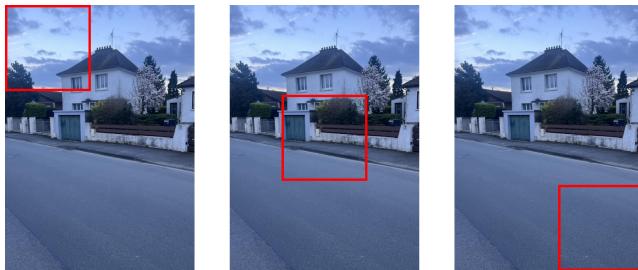


FIGURE 11 – Récupération d’images vides

également équilibrée en fonction de la classe la plus représentée.

2.3 Création du modèle décisionnel

2.3.1 Choix du modèle

La librairie scikit-learn propose plusieurs modèles pré-développés.

Voici un texte qui résume les avantages et inconvénients des algorithmes de classification pour la détection de panneaux de signalisation :

Dans le cadre de la détection de panneaux de signalisation, plusieurs algorithmes de classification peuvent être utilisés, chacun ayant ses propres avantages et inconvénients. Adaboost offre une bonne performance avec des données non-linéaires et améliore les modèles faibles, mais il est sensible au bruit et peut surapprendre avec trop d’itérations. Le K-Plus Proches Voisins (KNN) est simple à implémenter et fonctionne bien avec des données bien réparties, mais il peut être lent avec de grands ensembles de données et est sensible aux caractéristiques non pertinentes.

Les arbres de décision sont faciles à comprendre et à visualiser, fonctionnent avec des données catégorielles et numériques, et sont rapides à entraîner. Cependant, ils peuvent surapprendre et peuvent être instables avec de petites variations des données.

Les forêts aléatoires réduisent le surapprentissage par rapport aux arbres de décision individuels et offrent une bonne performance avec de grands ensembles de données.

Les SVM (Support Vector Machines) qui sont efficaces pour les espaces de grande dimension et offrent une bonne séparation des classes pour des frontières complexes. Toutefois, ils sont sensibles au choix du noyau et des hyperparamètres, et sont assez long à entraîner.

Les SVM et forêts aléatoires seront utilisées dans la

suite du rapport.

Chacun de ces modèles produit une sortie binaire par défaut. Pour prédire parmi plusieurs classes, la fonction OneVsRestClassifier a été utilisée. Cette stratégie consiste à ajuster un classifieur par classe. Pour chaque classifieur, la classe cible est ajustée contre toutes les autres classes. C’est la stratégie la plus couramment utilisée pour la classification multi-classes.

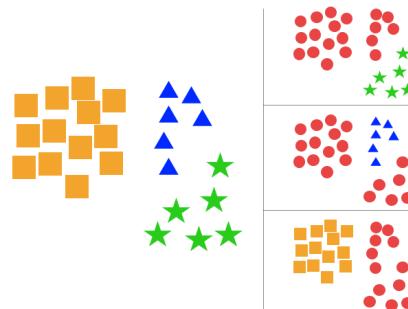


FIGURE 12 – Fonctionnement de OneVsRestClassifier

2.3.2 Un ou plusieurs classificateurs

Deux possibilités ont été envisagées :

- La première est la création d’un classifieur unique comportant les classes de panneaux et la classe “empty”.
- La deuxième était la création d’un classifieur pour repérer si la partie d’image est vide ou contient un panneau. Un autre qui classifie le type de panneau.

La première option a été choisie car la deuxième avait des bons résultats mais donnait beaucoup trop de faux positifs ce qui menait à une moins bonne performance générale.

	1 classifieur	2 classificateurs
Accuracy	0.72805	0.67274

TABLE 2 – Tableau de comparaison des choix

Après un réentrainement sur les faux positifs du jeu de données d’entraînement le classifieur a donné le résultat suivant :

classifieur après réentrainement
Accuracy

TABLE 3 – Tableau de comparaison des choix réentrainement

2.3.3 Entraînement du modèle

Avec tout cela, il est maintenant nécessaire de fournir les données au modèle afin de l'entraîner. Cependant, il reste quelques traitements à effectuer. Le premier est l'encodage de la classe. Il est possible d'utiliser un **LabelEncoder** qui va récupérer toutes les classes possibles : *Empty*, *Obligation*, *Stop*... et leur attribuer un nombre qui sera mieux compris par les modèles. *Empty* \Rightarrow 0, *Obligation* \Rightarrow 1, *Stop* \Rightarrow 2.

La fonction **StandardScaler** est ensuite appliquée sur les descripteurs. En général, les algorithmes apprennent mieux sur des données normalisées. Cela permet de passer de 67% d'accuracy à plus de 80%

Une **Analyse en composante principale** est utilisée afin de réduire les données et accélérer le traitement lors de l'entraînement et des prédictions des images.

Un calcul des poids de classe est également utilisée et inséré dans le modèle afin de rééquilibrer l'apprentissage de chaque classe. Ceci est notamment utilisé pour donner plus de poids à la classe vide afin d'éviter le plus de faux positifs.

Enfin la fonction **GridSearchCV** est utilisée pour trouver les meilleurs paramètres donnés au SVM et à la forêt aléatoire.

En utilisant les données de validations, il est possible d'obtenir :

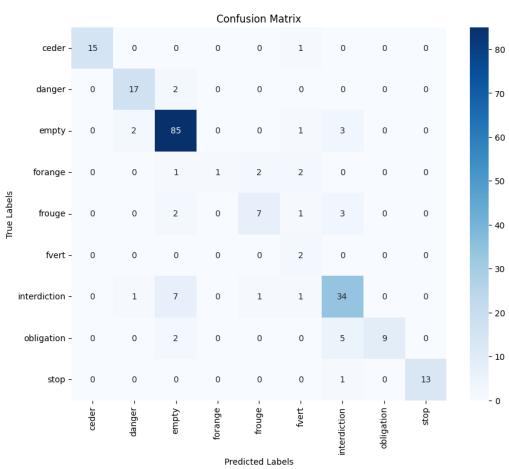


FIGURE 13 – Matrice de confusion

Validation Accuracy : 0.8280542986425339

- Classe **ceder** - Mean Average Precision : 0.9477
- Classe **danger** - Mean Average Precision : 0.9326
- Classe **empty** - Mean Average Precision : 0.9616
- Classe **forange** - Mean Average Precision : 0.4818
- Classe **frouge** - Mean Average Precision : 0.5678

- Classe **fvert** - Mean Average Precision : 0.3333
- Classe **interdiction** - Mean Average Precision : 0.7907
- Classe **obligation** - Mean Average Precision : 0.7178
- Classe **stop** - Mean Average Precision : 0.9461

Ce meilleurs modèle est le SVM avec les paramètres suivants :

Paramètre	Valeur
<i>C</i>	10
<i>gamma</i>	0.001
<i>kernel</i>	rbf

TABLE 4 – Paramètres optimaux du modèle SVM

Au travers de données de validation, il est possible de se rendre compte que le modèle marche plutôt correctement. Cependant, il n'arrive pas bien à reconnaître les différents types les feux tricolores. En effet, les feux verts, feux oranges et feux rouges ne diffèrent que très peu. De même, les interdictions et obligations étant des panneaux très similaires ne sont pas bien différenciés.

2.4 Détection de panneaux sur une nouvelle image

Après l'entraînement du modèle, la partie détection du projet a été entamée en utilisant l'algorithme de la fenêtre glissante. Pour améliorer l'efficacité de la détection, l'algorithme de pyramide d'images a également été appliqué. Enfin, les non-maxima ont été supprimés pour obtenir une détection plus précise.

2.4.1 L'algorithme de la fenêtre glissante

L'algorithme de la fenêtre glissante est une méthode courante de détection d'objets dans une image. Elle consiste à faire glisser une fenêtre de taille fixe, dans notre cas 64 x 64 pixels, sur l'image et à appliquer le modèle de classification préalablement entraîné à chaque emplacement de la fenêtre. L'algorithme est illustré en figure 14



FIGURE 14 – Exemple de fenêtre glissante

Cependant, l'utilisation d'une taille de fenêtre fixe dans cette méthode présente une limite : elle rend la détection d'objets de tailles différentes difficile. De plus, si un objet est plus grand que notre fenêtre, il sera également difficile de l'englober correctement. C'est pourquoi il est possible d'utiliser l'algorithme de pyramide en complément de la fenêtre glissante .

2.4.2 L'algorithme de pyramide d'images

L'algorithme de pyramide d'images est très pertinent pour notre étude, car les panneaux de signalisation dans notre base de données peuvent apparaître à diverses distances et tailles. Cette technique permet de détecter des objets de tailles variées en appliquant une fenêtre glissante à plusieurs échelles de l'image. L'algorithme commence par une fenêtre glissante de 64 x 64 pixels sur l'image originale, puis réduit progressivement la taille de l'image tout en conservant la taille de la fenêtre. Cette méthode détecte efficacement les petits objets en arrière-plan et les grands objets au premier plan, s'adaptant ainsi à la variabilité des tailles d'objets dans les images, ce qui est crucial pour une bonne détection.



FIGURE 15 – Exemple d'itérations de l'algorithme de pyramide d'images

Cependant, un nouveau défi doit-être traité : à chaque itération, une taille d'image différente est utilisée et les fenêtres glissantes se chevauchent. Il est donc nécessaire de trouver quelle fenêtre offre la meilleure détection des panneaux de signalisation.

2.4.3 L'algorithme de suppression des non-maxima

L'algorithme de suppression des non-maxima permet de pallier le problème de recouvrement des détections. Pour ce faire, une valeur de confiance est attribuée à chaque détection de panneau de signalisation en fonction de la probabilité d'appartenance à cette classe. C'est un score directement renvoyé du modèle en 2.3.3. L'algorithme de suppression des non-maxima conserve la valeur de confiance la plus élevée. Cet algorithme est très efficace, permettant d'obtenir une prédiction de *bounding box* plutôt précise.



FIGURE 16 – Exemple de suppression des non-maxima

2.4.4 Paramétrage des fonctions de détection

Pour améliorer la détection des panneaux dans l'image, plusieurs paramètres ont été utilisés :

Paramètre	Valeur
<i>confidence threshold</i>	0.9
<i>window</i>	64x64
<i>step</i>	16
<i>scale</i>	1.5
<i>overlap threshold</i>	0.3

TABLE 5 – Paramètres utilisés pour la détection des panneaux

2.5 Résultats de détection par machine learning

Beaucoup de modèles différents ont été essayés. Il n'était pas intéressant de venir sur chaque modèle car la plupart n'étaient même pas proche d'une détection intéressante. Seule la dernière itération présentée sur ce rapport se rapprochait d'une détection.

Il est possible d'observer la difficulté de la tâche en observant les détections sur l'ensemble de test. En règle général, les céder, danger, interdiction sont bien détectés. Cependant, les classes sous représentées comme les feux ne sont pas détectés.

Score	Valeur
<i>AP (sans classif)</i>	65.52
<i>AR (sans classif)</i>	15.19
<i>mAP (avec classif)</i>	17.19
<i>mAR (avec classif)</i>	6.37

TABLE 6 – Résultats Machine Learning

2.6 Pistes d'améliorations

Finalement, dans cette partie, le raisonnement accompli aboutit à des résultats relativement corrects.

Toutefois, il est possible d'améliorer encore la détection. Les performances actuelles pourraient être augmentées en intégrant des techniques avancées de *deep learning*, telles que les réseaux de neurones convolutifs (qui seront traités en partie 3), reconnus pour leur efficacité dans l'analyse d'images. De plus, la combinaison de cet algorithme avec des méthodes de segmentation d'image pourrait permettre une meilleure isolation des panneaux de signalisation du reste de la scène, améliorant ainsi la précision de leur localisation.

3 Détection d'objets avec Deep Learning

3.1 Enjeux du problème de détection

L'implémentation de la détection de panneaux de signalisation en deep learning est assez similaire à celle de la partie précédente. Cependant, deux approches ont été menées et seront présentées dans cette partie.

La première approche consiste à implémenter un réseau de neurones pour la tâche complète du projet, incluant la classification et la détection. Pour cela, l'idée était d'utiliser une architecture connue, comme celle du SSD (Single Shot Multibox Detector), et de modifier les couches de sortie afin de les adapter à notre problématique. Voici un schéma qui illustre cette approche.

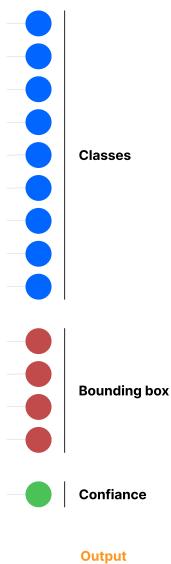


FIGURE 17 – Schéma de l'approche SSD

En effet, une sortie pour chaque classe a été affectée à savoir 9 dans notre cas, ainsi que quatre sorties pour chaque composante des coordonnées des bounding

boxes (voir figure 17). Enfin, une sortie a été associée à une valeur de confiance permettant de savoir à quel point le réseau est sûr de sa détection. Ce dernier paramètre est crucial, sinon nous nous retrouverions avec de nombreuses détections qui ne sont pas pertinentes. Pour les entrées, il s'agit de la même chose que précédemment, à l'exception de la nouvelle entrée du seuil de détection, où une valeur de 1 a été affectée aux fenêtres présentant un panneau, et 0 sinon.

Nous avons donné en entrée du réseau le résultat des sliding windows pour améliorer la performance du modèle car lui donner des images entière aurait été trop complexe à exploiter. Cependant, dans la partie précédente, l'utilisation de la sliding window seule ne permettait pas une détection efficace, car les panneaux présents au premier plan de l'image étaient répartis sur plusieurs fenêtres visible en figure 18. Cela s'avérait contre-productif, car nous entraînerions finalement notre réseau uniquement avec des panneaux présents en second ou arrière plan. Cependant, un test avec cette configuration a été réalisé.



FIGURE 18 – Exemple de problème d'un panneau en premier plan coupé par le sliding window

En effet, pour tenter de résoudre ce problème, une solution alternative a été envisagée. Plutôt que de se limiter aux fenêtres générées par la méthode de la fenêtre glissante, il aurait été préférable de prendre chaque boîte englobante et de l'ajuster de sorte que le panneau n'occupe pas toute la scène de la fenêtre, tout en s'assurant de n'avoir qu'un seul panneau par fenêtre. Cette approche permettrait de pallier le problème du manque d'entraînement des panneaux situés au premier plan des images. De plus, cette méthode favoriserait l'augmentation des données, car le jeu d'entraînement pourrait être quadruplé en divisant simplement les images en quatre. Cependant, cette méthode n'a pas pu être mise en œuvre par manque de temps.

Les résultats obtenus après une trentaine d'époques, présentés à la figure 9, montrent que le problème principal provient des prédictions des coordonnées des boîtes

Class accuracy	Confidence accuracy	Bbox mse
0.7924	0.7675	39451.3047

TABLE 7 – Résultat du SSD

englobantes. Ce résultat était attendu, car la prédiction des coordonnées des boîtes englobantes est généralement la plus difficile.

La seconde approche, celle que nous avons choisie, a consisté à procéder comme dans la partie précédente, en utilisant le réseau de neurones comme un classifieur multiquelques. La détection a été réalisée à l'aide des algorithmes mentionnés précédemment, à savoir la fenêtre glissante et la pyramide d'images.

3.2 Traitement du jeu de données

3.2.1 Dissimilarités avec la partie deep learning

En deep learning, le réseau de neurones nécessite un certain formalisme. En effet, cela implique que chaque image en entrée doit avoir des dimensions égales. Une bonne pratique consiste à normaliser les données entre 0 et 1 afin d'améliorer la robustesse du modèle. C'est l'objet de la partie suivante.

3.2.2 Normalisation de données

La normalisation des données est cruciale en deep learning. Elle consiste à normaliser les valeurs des pixels des images entre 0 et 1 en les divisant par 255, et à adapter les coordonnées des bounding boxes pour qu'elles correspondent à cette échelle normalisée.

En outre, les labels sont formatés via le one hot encoding. Ce principe transforme chaque label en un vecteur binaire, où un seul élément est à 1 (représentant la classe de l'image) et les autres à 0, facilitant ainsi le traitement par un réseau de neurones convolutionnel.

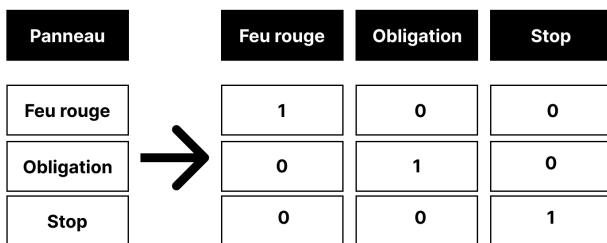


FIGURE 19 – Exemple du one hot encoding pour 3 classes

3.2.3 Ajout d'un padding

Finalement, la dernière étape du processus de prétraitement a été d'ajouter un **padding** à toutes les images, car les couches de convolution présentes dans le réseau réduisent la taille des images. Ainsi, pour que le réseau ne modifie pas la taille des images en sortie, une solution est d'ajouter un padding, c'est-à-dire d'élargir l'image pour ne pas réduire la taille des images en sortie du réseau. Pour ce faire, nous avons utilisé l'option `padding='same'` dans la création du modèle sur TensorFlow afin de ne pas avoir à le faire manuellement.



FIGURE 20 – Données prétraitées

3.3 Crédit du modèle de CNN

Pour réaliser le modèle, le réseau de neurones la bibliothèque TensorFlow a été utilisé. Tout d'abord le réseau ResNet a d'abord été implémenté. Comme demandé dans la consigne, la structure comprenant les couches a été réimplémentée sans utiliser un modèle pré-entraîné. Il a été nécessaire d'adapter les dimensions d'entrée du réseau, à savoir dans notre cas (224, 224, 3) pour les images et 9 classes pour les catégories.

Dans un premier temps, la fonction d'optimisation utilisée a été Adam, la fonction de perte a été la `categorical_crossentropy`, et enfin, la métrique de performance utilisée a été l'`accuracy`.

Afin d'assurer un entraînement de modèle efficace, la fonction `EarlyStopping` a été utilisée pour interrompre l'entraînement de manière opportune lorsque le réseau de neurones avait achevé son apprentissage, évitant ainsi le problème d'*overfitting* sur les données d'apprentissage. Cette fonction surveille une métrique spécifique, généralement la perte sur les données de validation, et arrête l'entraînement dès que cette métrique cesse de s'améliorer, ce qui permet de préserver le modèle de modifications ultérieures qui pourraient nuire à sa performance générale. Les données de validation ont également joué un rôle crucial en permettant de vérifier l'efficacité réelle de l'apprentissage.

En utilisant cette méthodologie, le `EarlyStopping` permet de trouver le meilleur compromis entre les performances et la fonction de perte, tout en tenant en compte les données de validation. Dans notre cas, le

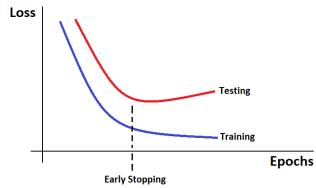


FIGURE 21 – EarlyStopping

meilleur compromis a été atteint après 19 époques. Voici le détail en figure 22 :

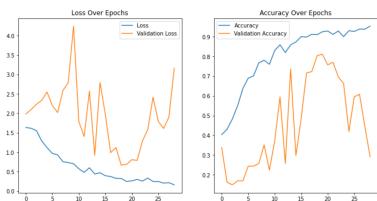


FIGURE 22 – Performence de l'apprentissage du CNN RestNet

3.4 Comparaison des performances des différents réseaux

Dans cette partie, nous avons souhaité comparer les différentes performances des réseaux uniquement sur le problème de classification des panneaux de signalisation à l'aide des bounding boxes, dans l'espoir d'obtenir de meilleurs résultats pour la détection. Il est vrai que cette liste n'est pas exhaustive.

Réseau	Accuracy	Val accuracy
ResNET50	98.59%	93.75%
VGG16	94.70%	95.63%
AlexNET	25.56%	27.50%
EfficientNET	35.52%	36.88%
MobileNET	84.25%	27.50%
Custom	46.15%	5.50%

TABLE 8 – Résultat des différents réseaux de neurones

Finalement, les réseaux ResNet-50 et VGG-16 sont les plus performants en termes de classification des panneaux de signalisation à l'aide de bounding boxes.

3.5 Résultats de détection par deep learning

Après avoir effectué la détection à l'aide des algorithmes mentionnés précédemment, les performances

obtenues n'ont pas répondu à nos attentes.



FIGURE 23 – Exemple de problème

En effet, ici par exemple, il est possible de remarquer que l'unique panneau présent dans cette image, le panneau d'obligation, est bien détecté mais malheureusement, sa probabilité d'appartenance à la classe n'est que de 0.96. De plus, un autre panneau d'obligation est détecté près du ciel avec un taux plus élevé de 0.99. Ainsi, même l'ajout d'une condition sur le seuil de confiance ou sur le nombre de panneaux que l'algorithme de détection doit identifier par image ne résoudrait pas le problème. La principale raison de cela doit provenir de la quantité limitée de données d'apprentissage et du nombre élevé de classes à comparer.

4 Résultats

Méthode	Machine learning	Deep learning
AP (sans classif)	65.52%	07.57%
AR (sans classif)	15.19%	00.52%
mAP (avec classif)	17.19%	01.02%
mAR (avec classif)	06.37%	06.37%

TABLE 9 – Comparaison des résultats

5 Conclusion

Pour conclure, la détection de panneaux est un défi complexe qui exige une quantité considérable de données d'apprentissage et l'utilisation de méthodes modernes, souvent gourmandes en mémoire et en ressources computationnelles.

Davantage de temps pour son développement et son optimisation, l'acquisition de données supplémentaires et l'utilisation du transfer learning et du fine-tuning pourrait significativement améliorer les performances de cette détection de panneaux.

Références

- [1] scikit-learn DOC Scikit-learn Disponible en ligne sur [ce lien](#).