

Projets SY32 — Détection de panneaux

Bartłomiej Grzadziel, Adrien Simon

06/2024

1 Introduction

1.1 Contexte

Le projet de cette année en *SY32* consiste à développer un modèle capable de reconnaître et de prédire différents types de panneaux à partir d'une image. Pour ce faire, avec l'ensemble des autres étudiants, nous avons d'abord réalisé un travail de préparation de la *data*. En effet, pour apprendre à un modèle à détecter des panneaux, on doit d'abord pouvoir lui fournir des images contenant des panneaux. C'est pourquoi nous sommes partis explorer les rues à la recherche de 5 panneaux différents et de feux de circulation.

Au total, nous avons dû fournir :

- 2 panneaux “dangers”
- 2 panneaux “obligation”
- 2 panneaux “interdiction”
- 2 panneaux “stop”
- 2 panneaux “céder le passage”
- 1 image pour chaque feu (“rouge”, “orange”, “vert”)
- 1 image sans panneau

Une fois les photos prises, nous les avons ensuite labellisées à l'aide de l'outil *labelimg* avec leurs noms associés. Nous avons dû aussi flouter des informations personnelles comme des plaques d'immatriculation et des visages. Enfin, nous avons envoyé ces images au professeur et avons pu constituer, à l'aide des autres étudiants, un *dataset* contenant près de 1000 images.



FIGURE 1 – Exemple de labelisation

1.2 Projet

Maintenant que les données sont prêtes, place à la partie *apprentissage*. Pour ce projet, nous avons à réaliser deux types d'apprentissages :

- L'un par méthode classique (*Adaboost*, *SVM*, *RandomForestClassifier*, etc.)
- L'autre par *réseaux de neurones*

Dans la suite de ce rendu, nous allons expliquer les différentes méthodes que nous avons utilisées pour l'apprentissage, et nous allons détailler les difficultés et solutions trouvées. Puis nous finirons sur une conclusion dans laquelle nous allons évoquer les potentielles améliorations possibles pour notre modèle.

2 Préparation de la data

Avant de passer à l'apprentissage, nous avons d'abord dû effectuer un travail conséquent de préparation de la data.

2.1 Augmentation et équilibrage du dataset

La première étape a été d'augmenter la taille du dataset et de l'équilibrer. En effet, en comptant chaque panneau pris en photo, on en sort avec ce résultat :

- Nombre de panneaux ‘frouge’ : 90
- Nombre de panneaux ‘ceder’ : 133
- Nombre de panneaux ‘interdiction’ : 365
- Nombre de panneaux ‘fvert’ : 109
- Nombre de panneaux ‘stop’ : 100
- Nombre de panneaux ‘danger’ : 162
- Nombre de panneaux ‘obligation’ : 108
- Nombre de panneaux ‘forange’ : 67
- Nombre de panneaux ‘ff’ : 10

On peut directement apercevoir un important déséquilibre entre le nombre de certains panneaux. On observe 289 panneaux contre 54 feux oranges, par exemple. Cela pose un problème puisque si nous faisons apprendre notre modèle sur un tel déséquilibre, notre

modèle sera super performant pour détecter des panneaux *interdiction* et très mauvais pour détecter des feux oranges.

Pour répondre à ce problème, nous avons écrit un script qui sélectionne les panneaux les moins présents dans le *dataset* (par exemple les feux) et duplique l'image en la retournant et en y ajoutant un bruit. Ainsi, non seulement on équilibre notre *dataset*, mais on fournit davantage d'images pour l'apprentissage.

Voici un exemple illustré avec l'image 0001.jpg dupliquée (voir Figure 2).



(a) Image originale



(b) Image dupliquée

FIGURE 2 – Comparaison des images originale et dupliquée

Une fois ce travail effectué, nous avons aussi dû dupliquer les fichiers *CSV* associés à ces images en mettant à jour proprement l'emplacement du nouveau panneau sur l'image afin que la boîte *labeling* corresponde aux coordonnées du nouveau panneau.

Enfin, nous avons dû traiter les cas des panneaux *stop* à part, étant donné que leur renversement vers le côté produisait un apprentissage sur un panneau assez différent dû au texte "STOP" présent dessus, comme illustré sur la Figure 3 :



FIGURE 3 – Panneau stop inversé

2.2 Ajout de labels

La deuxième partie de la préparation de la *data* a été d'ajouter un label "*empty*" à toutes les images ne contenant pas de panneaux. En effet, afin d'entraîner notre modèle et lui apprendre ce qui ne correspond pas à un panneau, nous avons dû écrire dans les fichiers *CSV* des images sans panneaux un texte du style "X,X,X,X,*empty*".

Ainsi, avec l'ajout des labels et la duplication de certaines images, nous obtenons un *dataset* bien plus équilibré et nous pouvons enfin entrer dans la phase d'apprentissage.

Dataset après équilibrage :

- Nombre de panneaux 'frouge' : 153
- Nombre de panneaux 'ceder' : 223
- Nombre de panneaux 'interdiction' : 365
- Nombre de panneaux 'fvert' : 170
- Nombre de panneaux 'stop' : 168
- Nombre de panneaux 'danger' : 252
- Nombre de panneaux 'obligation' : 195
- Nombre de panneaux 'empty' : 54
- Nombre de panneaux 'forange' : 104
- Nombre de panneaux 'ff' : 10

On remarque effectivement que le nombre d'images labellisées "empty" est assez faible par rapport aux autres catégories d'images. Pour équilibrer l'apprentissage, il n'est cette fois-ci pas nécessaire de dupliquer les images sans panneau. En effet, il suffit de diviser chaque image sans panneau en quatre sous-images lors de l'apprentissage et d'assigner à chacune d'elles la classe "empty" comme montré sur la figure 4.



L'objectif était d'apprendre à distinguer efficacement les zones sans panneaux des images comportant des panneaux. Cependant, les résultats obtenus n'étaient pas satisfaisants.

Pour améliorer la performance du modèle, nous avons adopté une approche plus ambitieuse : chaque image "empty" est désormais divisée en 30 sous-images distinctes. Cela a considérablement augmenté la quantité de données d'apprentissage, permettant au classifieur de mieux généraliser et de détecter plus efficacement les zones sans panneaux dans de nouvelles images. Cette méthode a nettement amélioré nos résultats de détection, comme nous le verrons par la suite.

3 Apprentissage

3.1 Quelques erreurs

Pour l'apprentissage de notre modèle, nous avons adopté différentes approches. Nous avons d'abord choisi un modèle d'apprentissage vu en cours. Une fois le modèle établi, nous avons tenté de lui faire apprendre en lui fournissant des images entières et en utilisant les étiquettes. Par exemple, une image étiquetée « feu rouge » devait apprendre au modèle qu'elle contient effectivement un feu rouge (voir Figure 5).



FIGURE 5 – Illustration d'un mauvais apprentissage

Une fois l'apprentissage terminé, nous avons évalué notre modèle sur le jeu de données de validation, mais les résultats ont été très décevants, avec seulement 22 % de bonnes prédictions.

Face à cette performance médiocre, nous avons analysé notre méthode pour identifier le problème. Après réflexion, nous avons réalisé que notre approche d'apprentissage était inadaptée. En effet, nous avions appris à notre modèle que toute image contenant un panneau feu rouge, y compris celles avec le ciel et les bâtiments, correspondait à un feu rouge. La solution était en fait de

limiter l'apprentissage aux coordonnées fournies dans le fichier *CSV* des étiquettes. Ainsi, après cette correction, notre modèle a été entraîné à reconnaître les descripteurs spécifiques extraits des emplacements précis des panneaux sur les images.

Grâce à cette méthode d'apprentissage plus précise, nous avons amélioré notre taux de détection de 22 % à 91.2 %, ce qui est un résultat très satisfaisant.

3.2 Choix des descripteurs & du classifieur

Le choix des descripteurs et de la méthode d'apprentissage est crucial pour obtenir de bons résultats en traitement d'images. Dans notre cas, nous avons procédé par tâtonnements pour trouver la combinaison optimale.

Initialement, nous avons utilisé un classificateur de *forêt aléatoire* combiné à des descripteurs *SIFT* (*Scale-Invariant Feature Transform*). Cependant, les résultats n'étaient pas satisfaisants. Nous avons alors testé la méthode de classification *SVM* (*Support Vector Machine*) avec des descripteurs *HOG* (*Histogram of Oriented Gradients*), et les résultats étaient si encourageants que nous avons décidé de nous concentrer sur cette combinaison.

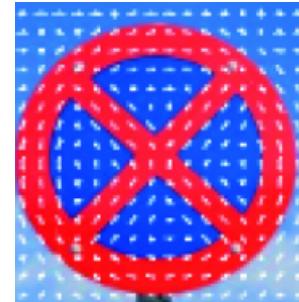


FIGURE 6 – Illustration du descripteur HOG

Nos recherches ultérieures ont confirmé que ce choix était judicieux. En effet, les descripteurs *HOG* sont particulièrement efficaces pour extraire des formes claires avec des contours bien visibles. Cela est dû au fait que les descripteurs *HOG* calculent l'orientation et la magnitude des gradients dans une région de l'image, puis construisent un histogramme de ces orientations. Comme visible sur la figure 6, la nette variation des gradients permet d'identifier des formes distinctes, telles que la croix dans cet exemple.

Les descripteurs *HOG* associés à la méthode *SVM* ont contribué à ce meilleur résultat, puisqu'ils sont par-

ticulièremment efficaces pour prédire des formes simples, telles que celles des panneaux. La méthode *SVM* est une technique d'apprentissage supervisé qui permet de trouver un hyperplan séparant deux classes de données. Dans notre cas, les données sont les descripteurs *HOG* des panneaux, et avant de prédire le type de panneaux, les "deux classes" sont "panneau présent" et "panneau absent". Cela nous aidera à détecter plus facilement des panneaux dans la partie détection.

4 Détection

4.1 Quelques idées

4.1.1 Carrés aléatoires

Une fois que nous avons bien entraîné notre modèle avec un taux de prédiction élevé en lui fournissant les coordonnées exactes du panneau dans l'image, nous devons maintenant passer à l'étape suivante : la détection.

Notre première idée a été de placer des carrés de manière aléatoire sur l'image et de tenter de prédire si oui ou non ces carrés contenaient un panneau. Cependant, après seulement deux tests, nous avons rapidement réalisé que cette méthode était peu efficace (voir Figure 7).

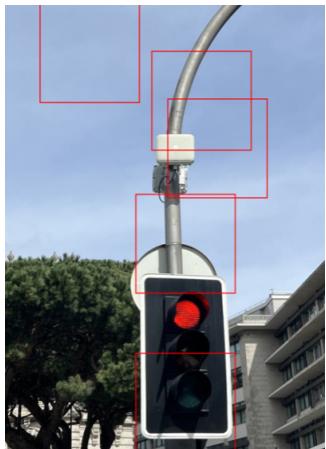


FIGURE 7 – Détection aléatoire

Effectivement, le fait de placer les carrés de manière aléatoire sur l'image présentait plusieurs inconvénients. Tout d'abord, les carrés ne recouvriraient pas systématiquement toute l'image, ce qui réduisait les chances de détecter un panneau.

De plus, notre modèle était peu efficace pour bien détecter le type de feu de circulation, car un carré ne

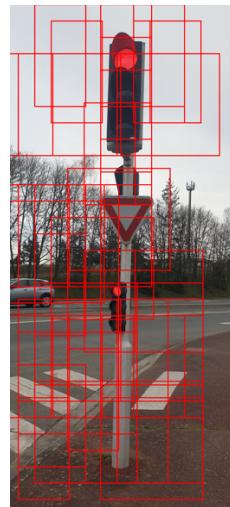


FIGURE 8 – Détection aléatoire V2

recouvrait que rarement un feu en entier. Cela rendait la tâche de classification plus difficile pour le modèle.

C'est pourquoi nous avons implémenté une version améliorée de cette détection en plaçant davantage de carrés sur l'image mais aussi en y plaçant des rectangles verticaux pour mieux reconnaître la forme des feux .

Les résultats ont été légèrement meilleurs mais restent peu convaincants. Sur les trois panneaux présents sur la figure 8, seul un feu rouge a été correctement détecté. De plus, en raison de la similarité entre les panneaux de danger, d'interdiction et de céder le passage, et vu que le détecteur a du mal à se positionner idéalement sur le panneau, le classifieur classe parfois le panneau céder le passage comme un panneau d'interdiction, et parfois comme un panneau de danger.

4.1.2 Reconnaissance de forme

Une deuxième approche que nous avons envisagée a été d'utiliser des bibliothèques *Python* pour essayer de détecter des formes géométriques telles que des carrés, des rectangles et des cercles sur l'image, puis d'effectuer la détection de panneaux et de feux de circulation sur ces éléments.

Cependant, cette méthode qui semblait prometteuse sur le papier n'a malheureusement pas donné de résultats très convaincants. En effet, la détection de formes géométriques sur une image est une tâche complexe qui peut être affectée par de nombreux facteurs tels que l'éclairage, la perspective, la présence de bruit, etc.

Comme nous pouvons le voir sur la figure 9, la détection des formes est assez aléatoire. C'est à nouveau le problème des feux qui pose principalement problème.

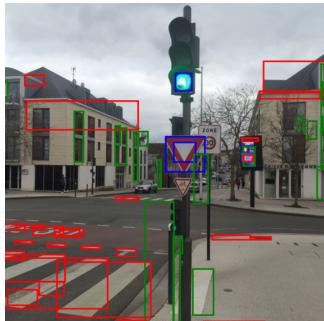


FIGURE 9 – Reconnaissance des formes

En effet, nous n'arrivons pas à faire reconnaître un rectangle vertical sur l'image et donc la détection d'un feu sur une image reste assez mauvaise. Cependant, cette méthode s'avère plutôt efficace pour la détection des autres panneaux. Par exemple, sur la Figure 9, l'algorithme a très bien détecté le panneau céder et la classification a été très bonne (95% de bonnes classifications lorsque le panneau est bien détecté).

4.1.3 Fenêtre glissante et pyramides d'images

Après ces deux échecs, nous avons décidé d'appliquer différentes méthodes vues en cours afin de les combiner pour obtenir un détecteur assez efficace.

D'abord, nous appliquons le principe de la fenêtre glissante sur toute l'image. Nous avons décidé d'utiliser des carrés de 64x64 pixels pour détecter les panneaux de signalisation qui ne sont pas des feux tricolores. En effet, étant donné la forme particulière des feux tricolores, nous avons privilégié des fenêtres différentes, de forme rectangulaire verticale de 64x128 pixels, afin de mieux capturer la forme des feux sur une image.

Ainsi, nous parcourons toute l'image et stockons pour chaque carré quel panneau a la plus grande probabilité de prédiction. Nous stockons cette valeur dans un tableau qui nous servira plus tard pour récupérer le panneau qui a été détecté avec la plus grande probabilité.

La deuxième étape est d'utiliser la *pyramide d'images* vue en cours. Nous appliquons une fenêtre glissante qui parcourt toute l'image sur différentes tailles. Nous avons choisi de diviser la taille de l'image quatre fois pour obtenir un maximum de données pour la détection ensuite. Une fois que nous avons itéré quatre fois ce processus sur l'image, nous récupérons les carrés avec le plus grand pourcentage de prédiction. Si un carré indique une prédiction à plus de 75% pour un panneau sur l'image, alors nous considérons que ce panneau est présent sur l'image.

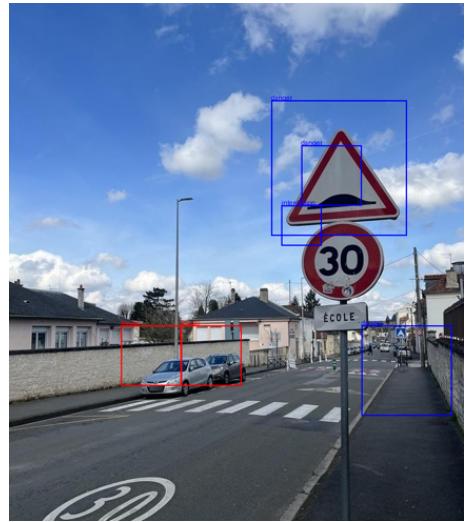


FIGURE 10 – Résultats de la fenêtre glissante et de la pyramide d'images

L'étape suivante consiste à récupérer toutes les prédictions effectuées par chacune des fenêtres glissantes sur chaque taille d'image et à comparer les résultats. Les fenêtres ayant détecté un panneau avec la plus grande précision sont récupérées et comparées à celles effectuées sur des images de différentes tailles.

Comme nous pouvons le voir sur la Figure 10, les résultats obtenus par cette méthode laissent à désirer. Ce sont des résultats obtenus après avoir supprimé toutes les prédictions à moins de 70%. Certaines prédictions de panneaux sont correctes, comme le panneau de danger qui a été détecté par deux carrés sur deux tailles d'images différentes. En revanche, ce n'est pas le cas pour les autres detections.

Tout d'abord, le code a détecté deux feux rouges dans le vide sans raison apparente. De plus, un panneau de danger a été détecté en bas à droite de l'image alors que le carré entoure simplement un pavé pour piétons. Enfin, le panneau d'interdiction a été très mal détecté puisque seulement une petite partie de celui-ci a été identifiée, et les coordonnées de la détection ne rentrent pas dans les 50% de marge autorisées par l'*IOU* (*Intersection over Union*).

Une première piste d'amélioration a été d'effectuer un tri pour supprimer les doublons (les carrés qui englobent d'autres carrés), choisir les meilleures prédictions et supprimer les carrés adjacents. En effet, si deux carrés ont détecté le même panneau sans s'entremêler à moins de 20 pixels, nous considérons alors qu'ils ont détecté le même panneau.



FIGURE 11 – Détection trop proche

La Figure 11 illustre le problème de la "détection trop proche" auquel nous faisions face. Le code a bien détecté que les deux éléments entourés correspondaient à des panneaux d'interdiction, mais a renvoyé deux cases prédites différentes. Nous avons alors décidé de fusionner les deux boîtes pour en créer une plus grande lorsque cela se produit, afin d'augmenter les chances de couvrir la zone réelle du panneau et d'améliorer la détection.

4.2 Solution adoptée

Malheureusement, cette méthode n'est pas assez robuste et a souvent du mal à détecter tous les panneaux présents sur l'image. De plus, il arrive fréquemment que des carrés soient détectés dans le ciel. Par ailleurs, si aucun carré sur l'image n'a détecté un panneau avec une précision supérieure à 85%, nous considérons alors que l'image ne contient pas de panneaux.

Pour consolider notre détecteur, nous avons recours à notre détecteur de formes mentionné précédemment. Comme nous l'avons précisé, le détecteur de formes est très efficace pour détecter des panneaux ronds et triangulaires et renvoie ainsi les coordonnées exactes de la boîte correspondant à ce panneau. Ensuite, notre classifieur, étant le plus performant lorsqu'il dispose des coordonnées exactes du panneau en entrée, réalise généralement la classification à la perfection. Cependant, cette méthode de détection présente des limites, car elle détecte des formes un peu partout. C'est pourquoi nous avons ajouté un seuil de détection de 90% pour ne renvoyer que les panneaux pour lesquels l'algorithme est quasiment certain de la détection.

Ainsi, nous avons intégré dans notre algorithme de détection basé sur des fenêtres glissantes et des pyramides d'images, la détection de formes pour consolider les résultats. Nous avons pu procéder à une première évaluation de notre classifieur et détecteur sur l'ensemble du fichier "val" contenant les images pour l'évaluation. Après exécution du code, notre classifieur et détecteur ont réalisé une performance de 40% de bonnes prédi-



FIGURE 12 – Complexité de la détection d'un objet

tions, ce qui est décevant car cela est inférieur à un modèle aléatoire.

C'est à ce moment-là que nous nous sommes souvenus de l'image présente dans le cours (Figure 12) expliquant la complexité de la détection d'un objet.

Nous avons enfin pu comprendre le message de l'image par notre propre expérience.

4.3 Raison de la faiblesse du modèle

La principale faiblesse de notre modèle provient principalement du détecteur, mais aussi du classifieur. Le problème du classifieur est qu'il n'a pas été assez entraîné. Nous trouvons que la base d'images proposée pour l'apprentissage est trop petite. Bien que nous ayons essayé de l'agrandir par différentes méthodes, elle reste assez limitée (par exemple, 104 feux oranges pour l'apprentissage n'est clairement pas suffisant). C'est pourquoi notre code détecte souvent des panneaux dans le ciel ou sur des éléments qui n'en sont pas, car il n'a pas pu suffisamment apprendre.

La deuxième raison est due à la nature des panneaux à détecter. En effet, bien que nous ayons environ 92% de bonnes prédictions lorsque nous nous situons exactement sur les coordonnées du panneau, la plupart du temps, la fenêtre glissante (*sliding window*) tombe sur une partie du panneau à détecter, et très rarement elle réussit à capturer exactement les coordonnées du panneau. Ce problème entraîne une mauvaise classification, car nous avons de nombreux panneaux qui se ressemblent beaucoup s'ils sont mal détectés.

Nous pouvons assez bien illustrer ce problème avec les panneaux "céder le passage" et "danger", mais aussi avec les différents feux. Ainsi, si la fenêtre glissante (*slid-*

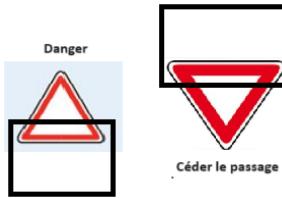


FIGURE 13 – Difficulté de prédiction entre Danger et Céder

ding window) atterrit mal sur un panneau "Danger", notre modèle peut croire qu'il s'agit d'un panneau "céder le passage" car la forme est assez similaire. De même pour l'inverse, très souvent notre modèle reconnaît bien un panneau sur une image, mais classait assez mal et confondait souvent les panneaux "Danger" et "céder le passage".

Un exemple encore plus flagrant concerne les feux de circulation. En effet, il est très rare qu'une fenêtre glissante atterrisse exactement sur les coordonnées d'un feu. La quasi-totalité du temps, dès que notre modèle détectait un feu sur une image, il se trompait sur le type de feu. Le code renvoie la plupart du temps "*fvert*" (feu vert) car il est présent en plus grand nombre dans le *dataset* d'apprentissage, et il a beaucoup plus de mal à détecter le "*forange*" (feu orange) qui est bien plus rare.

4.3.1 Comment régler ce problème ?

Bien que nous ayons implémenté une fonction calculant les non-maxima pour déterminer l'aire de recouvrement entre les fenêtres, les résultats n'ont pas été satisfaisants. Nous avons peut-être mal implanté le calcul du recouvrement entre les différentes échelles de l'image, ce qui pourrait produire de mauvais résultats pour cette méthode.

Une autre raison pourrait être liée au classifieur, qui a seulement appris à reconnaître les panneaux lorsqu'il dispose exactement des coordonnées du panneau. Il est donc moins performant pour reconnaître un panneau lorsqu'on lui montre uniquement une petite partie de celui-ci.

Une solution serait de reprendre notre *dataset* d'apprentissage et de faire davantage apprendre notre modèle sur les mêmes images, mais en décalant légèrement les boîtes de coordonnées. Prenons l'exemple d'un panneau de danger (voir Figure 14).



FIGURE 14 – Amélioration de l'apprentissage pour la détection des panneaux

Sur cet exemple, un panneau de danger est découpé en trois parties différentes, mais assez grandes pour que le classifieur ne puisse pas les confondre avec d'autres panneaux. En divisant ainsi les panneaux pour l'apprentissage, nous aurions un classifieur plus apte à détecter les panneaux lorsque la fenêtre glissante ne tombe pas idéalement sur le panneau et qui serait potentiellement plus performant. Ce classifieur serait donc plus robuste, car il pourrait reconnaître des panneaux dans des situations plus complexes.

Nous sommes conscient que le rôle de la non-max suppression est justement de palier à ce problème, mais nous émettons simplement une hypothèse comme quoi cette méthode pourrait potentiellement aider dans la détection des panneaux. Bien que cette méthode présente aussi des défauts puisque si on réussit bien à reconnaître une partie du panneau, on ne pourra pas récupérer les coordonnées exactes du panneau.

5 Apprentissage Profond

Dans cette section, nous présentons notre approche basée sur l'apprentissage profond pour la classification d'images. L'apprentissage profond permet de construire des modèles plus performants en apprenant directement à partir des données brutes, sans nécessiter d'extraction manuelle des caractéristiques via des HOGs ou des Bag of Words. Nous avons choisi d'utiliser un réseau de neurones convolutifs (CNN) pour cette tâche, en raison de la capacité de ce type de réseau à capturer des motifs complexes dans les images.

5.1 Première approche

La première approche pour approcher la tâche avec de l'apprentissage profond a été de réaliser la même chose que précédemment : créer un classifieur entraîné sur les bounding boxes, et puis faire la détection via une fenêtre glissante.

La première étape a donc été d'implémenter un réseau de neurone spécialisé dans la classification. Pour

faire cela nous avons utilisé la librairie Tensorflow. Des essais avaient été menés avec Pytorch mais nous avons trouvé cette dernière plus difficile d'utilisation, et moins efficace sur nos tâches spécifiquement.

Ce réseau est composé de 3 couches de convolution, séparée par du pooling, puis de deux couches complètement connectées.

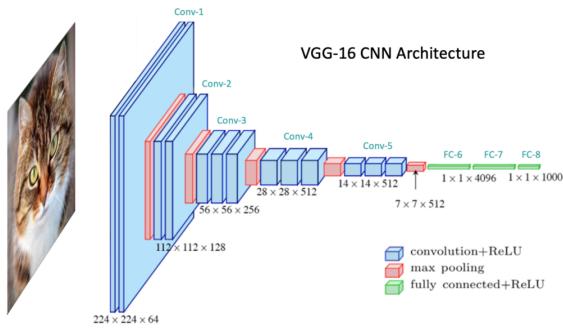


FIGURE 15 – Exemple de réseau de neurone convolutif à 5 couches de convolution.

Afin d'améliorer les performances et la vitesse d'apprentissage du modèle, plusieurs optimisations ont été mises en œuvre :

- Batching : Les images sont traitées par batch de 32, ce qui permet d'améliorer la stabilité de l'apprentissage et d'accélérer la convergence du modèle.
- Optimiseur : Nous avons utilisé l'optimiseur Adam, pour ses performances et sa capacité à ajuster dynamiquement les taux d'apprentissage, ce qui a aidé à obtenir des résultats optimaux plus rapidement.
- Fonction de Loss : La fonction de perte Cross-Entropy a été utilisée, ce qui est standard pour les tâches de classification et aide à améliorer la précision du modèle.

Ce classifieur présente de bons résultats en classification (97%) sur des restrictions des images à leurs bounding boxes. Cependant, pour la détection la tâche était moins facile. En effet, avec cette approche, nous avons fait face aux mêmes problèmes que précédemment : panneaux détectés dans le ciel, certains feux mal détectés, etc.

Faisant face aux mêmes problèmes que précédemment, nous nous sommes décidés à tenter une approche de détection également par réseau de neurones.

5.2 Détection par réseau de neurones

Pour construire un réseau de neurones qui cherche à faire de la détection, il faut modifier la sortie du réseau de neurone. Le but est ici d'avoir en sortie un vecteur

comportant les coordonnées de la boîte, de la forme (x_1 , y_1 , x_2 , y_2). La sortie doit donc être 4 nombres entiers (ou flottants, puis arrondis), afin de déterminer les délimitations des zones d'intérêts sur les images.

Cette approche de déterminer les bounding boxes directement via un réseau de neurone est utilisée par des modèles spécialisés en Computer Vision, et nous a paru prometteuse, elle est cependant difficile à mettre en place.

En effet, le réseau que l'on a mis en place considère qu'une boîte englobante labelisée 'empty', n'est en fait pas vide. C'est une boîte 'empty' au même titre qu'une boîte 'obligation' ou 'frouge'. Il a donc fallu s'attarder sur le traitement de ce cas.

La solution à laquelle nous avons pensé a été de supprimer totalement les boîtes vides du jeu d'apprentissage, ce qui permet au réseau de n'apprendre à localiser que les boîtes présentant un intérêt réel.

L'objectif de ce réseau est de discriminer les zones d'intérêt d'une image complète. Cependant, cette approche présente deux inconvénients majeurs :

- Détection limitée : Le réseau ne retourne qu'une seule bounding box par image, alors que certaines images peuvent en contenir plusieurs.
- Performance et convergence : Le réseau éprouve des difficultés à discerner correctement les zones d'intérêt. La perte ne diminuait plus après cinq époques d'apprentissage et restait excessivement élevée. Cela suggère que le réseau n'est pas assez complexe et qu'il n'arrive pas à généraliser correctement ses résultats.

Ceci se comprend assez simplement, en effet en observant des réseaux comme ceux d'architectures telles que Yolo, SSD, ou Faster-R-CNN, notre architecture est bien trop simple pour espérer avoir des performances proches de ces derniers.

Cette approche a donc été abandonnée pour la suite de nos recherches, et nous avons décidé d'optimiser la première tentative.

5.3 Nouvelle méthode de classification

Pour tenter d'améliorer une dernière fois le modèle, nous avons tenté une nouvelle approche pour la classification. Nous pensons que la méthode par fenêtre glissante fonctionne, mais qu'elle n'est pas assez bien exploitée. Notre but est donc de trouver un moyen de détecter les fenêtres intéressantes, et ensuite, de prédire ce qu'il se trouve sur les fenêtres en question. Pour cela, nous nous y sommes pris en 3 étapes.

5.3.1 Création d'une labellisation alternative.

Le but de notre nouvelle approche est de prédire si une zone donnée comprend ou non un intérêt. Pour cela, nous avons créé une nouvelle labellisation des données en deux classes : `non_empty` et `empty`. Les bounding boxes comprenant des panneaux et des feux ont donc été re-labellisées en `non_empty`. Afin d'équilibrer le jeu de donnée, des nouvelles bounding boxes, vides, ont été créées sur chaque image, et labellisées `empty`. Pour faire cela nous avons fait un script qui génère entre 2 et 4 boxes par image, sur des zones n'ayant aucun recouvrement avec des boxes déjà existantes. Ceci permet, en supposant la bonne labellisation initiale de chaque image, de s'assurer que les boxes sont bien vides.

Cette approche nous a permis d'obtenir un jeu de données à deux classes, vides ou non-vides, assez équilibré, qui va nous permettre de discriminer les zones intéressantes des zones inutiles.

5.3.2 Création d'un deuxième réseau de classification.

Un nouveau réseau de neurone a donc été créé, similaire au premier, mais lui destiné à prédire si une zone contient quelque chose ou non.

Pour l'entraînement, les images ont à nouveau été restreintes aux bounding boxes, pour chaque boîte de la nouvelle labellisation, et le réseau a été entraîné.



FIGURE 16 – Exemple de labellisation d'images.

Ce réseau simple s'est révélé efficace à 98%, ce qui nous a paru satisfaisant pour continuer. Ce réseau sera par la suite désigné comme Réseau de Discrimination, à opposé au Réseau de Classification qui est le réseau initial.

5.3.3 Utilisation des deux réseaux

La méthode utilisée a donc été la suivante : - Première étape : Faire une fenêtre glissante sur différentes tailles d'images, et prédire le contenu des fenêtres avec le réseau de discrimination.

- Deuxième étape : Ne garder que les fenêtres labellisées `non_empty`.

- Troisième étape : Prédire toutes les fenêtres non-vides avec le réseau de classification.

- Quatrième étape : Utiliser un algorithme de suppression des non-maxima, et de regroupement avec le calcul d'aire de recouvrement, pour détecter les aires d'intérêt optimales.



FIGURE 17 – Exemple de détection.

On peut voir sur cet essai, avec une image du jeu de validation, que le feu rouge, et seul le feu rouge, est détecté. Nous avons trouvé cette approche prometteuse, et potentiellement plus efficace que la première approche de classification par réseau de neurone. Nous avons donc décidé d'essayer celle-ci pour prédire les données du jeu de test.

6 Performances

6.1 Analyse des Résultats pour la méthode classique

Métrique	Valeur
AP (sans classification)	31.46%
AR (sans classification)	25.60%
mAP (avec classification)	18.26%
mAR (avec classification)	30.45%

TABLE 1 – Performances du Modèle (Classique) sur le dataset "val"

Comme nous pouvons l'observer sur le tableau ci-dessus, notre modèle obtient des performances assez mauvaises. Effectuons une courte analyse des résultats :

La précision moyenne (AP) sans tenir compte des classes est relativement basse à 31.46. Cela veut dire que le modèle a une faible capacité à identifier correctement les objets sans erreur de classification. De plus, le rappel moyen (AR) sans tenir compte des classes est également faible, à 25.60, ce qui indique que le modèle manque la plupart des panneaux présents sur l'image et qu'il parvient uniquement à en détecter le quart.

Pour ce qui est des résultats avec classification, la précision moyenne (mAP) est encore plus basse, à 18.26. Ce résultat démontre simplement que lorsque le modèle doit non seulement détecter un panneau mais aussi le classer correctement, il devient encore moins précis.

Cependant, le rappel moyen (mAR) en tenant compte des classes est légèrement plus élevé que l'AR sans classification, à 30.45. On se doute alors que le modèle est un peu meilleur pour détecter les panneaux lorsqu'il tient compte des classes, même s'il ne les classe pas toujours correctement.

Les performances globales du modèle sont sous-optimales, avec des scores de précision et de rappel relativement bas. Nous sommes assez déçus de ces résultats compte tenu du temps investi dans ce projet. Comme mentionné précédemment, le principal problème réside dans la détection, car notre classification est très performante (92%) lorsque nous disposons des coordonnées exactes du panneau.

Bien que nous ayons essayé différentes méthodes de détection, en pensant suivre la meilleure approche pour améliorer ce processus, et en combinant intelligemment différentes méthodes afin de ne pas obtenir un détecteur trop lent, nous n'avons pas réussi à créer un détecteur solide. Notre objectif était de développer un détecteur rapide et peu coûteux, mais les résultats n'ont pas été à la hauteur de nos attentes.

Métrique	Valeur
AP (sans classification)	29.84%
AR (sans classification)	20.79%
mAP (avec classification)	10.32%
mAR (avec classification)	25.12%

TABLE 2 – Performances du Modèle (Classique) sur le dataset "test"

6.2 Analyse des Résultats avec l'apprentissage profond.

Les résultats, comme il est possible de le voir sur le tableau 3, ne sont pas bons. Au vu des performances de ce modèle, on peut en déduire que la détection ne

Métrique	Valeur
AP (sans classification)	00.04%
AR (sans classification)	00.13%
mAP (avec classification)	00.02%
mAR (avec classification)	00.71%

TABLE 3 – Performances du Modèle Profond sur le dataset "test"

fonctionne clairement pas.

Le très faible score d'AP indique que le modèle rate la majorité des objets présents dans les données d'évaluation. Le score de mAP montre lui que le modèle a une capacité quasiment nulle à détecter et classer les objets correctement. C'est à dire que même lorsqu'un objet est détecté, il n'est pas forcément bien classifié, ce qui est assez perturbant au vu des performances initiales du modèle de classification.

La dernière approche avec le modèle de discrimination, tentée assez tard dans le projet, n'a pas eu le temps d'être poussée et paye le manque de temps d'entraînement des modèles, et de finetuning des paramètres de l'algorithme de fenêtre glissante. En effet, avec toutes ces prédictions à faire, avec deux réseaux de neurones, un cycle complet de prédiction sur le jeu de test prend 40 minutes.

L'approche présente à première vue deux problèmes :

- Le réseau de discrimination semble avoir fait du sur-apprentissage sur le jeu d'entraînement. En effet, il présentait de très bonnes performances sur les jeux train et val, mais détecte bien trop de fenêtres, qui plus est des fenêtres vides, sur une seule image. Des panneaux sont par exemple toujours détectés dans le ciel.
- La fenêtre glissante a mal été implémentée, et aurait pu être bien plus efficace.

La dernière idée représentait de bons espoirs pour nous, mais nous avons malheureusement manqué de temps et nous aurions pu pousser cette approche plus loin si nous l'avions trouvée plus tôt.

7 Axes d'amélioration

7.1 Apprentissage Classique

Bien que nous ne présentions pas les meilleurs résultats, nos performances ont augmenté de près de 12% depuis le premier essai jusqu'au dernier (de 19% de précision initialement à 31% à la fin). Cela est un résultat encourageant qui prouve que si nous avions davantage de temps pour tester différents paramètres, nous au-

rions peut-être éventuellement fini par trouver un bon modèle capable de correctement prédire les panneaux sur les images.

Pour obtenir de meilleurs résultats, nous aurions pu entraîner un classifieur plus performant sur un ensemble de données plus large en utilisant davantage de techniques pour augmenter la taille du dataset, ainsi qu'en ajustant les paramètres du classifieur. Par exemple, augmenter le nombre de descripteurs utilisés par HOG aurait pu être une solution bénéfique, malgré le coût supplémentaire en temps.

Actuellement, notre modèle a encore des difficultés à distinguer certains panneaux qui se ressemblent, en particulier pour différencier les feux tricolores entre eux, qui ne se distinguent que par une boule de lumière positionnée différemment selon le feu. Un problème potentiel est que la non-max suppression peut parfois sélectionner une fenêtre qui prédit mieux un feu rouge alors que l'image montre un feu orange. Augmenter le nombre de descripteurs et le nombre de feux tricolores pourrait éventuellement régler ce problème.

Enfin, notre modèle prend environ 10 minutes pour prédire tous les labels des images actuellement. Si nous améliorons ses performances, la classification pourrait prendre certes plus de temps, mais les résultats seront de meilleure qualité.

7.2 Partie CNN

Pour améliorer la dernière approche, il aurait fallu pouvoir améliorer l'algorithme de fenêtre glissante. L'idée de le faire tourner plusieurs fois avec des fenêtres carrées pour les panneaux, et rectangles pour les feux aurait permis de pouvoir gagner en précision sur les deux types d'objets à détecter.

Nous pensons réellement que cette approche était prometteuse, mais qu'elle a simplement manqué de temps pour être améliorée et approfondie.

Pour avoir de meilleurs résultats, il aurait peut-être fallu s'inspirer de modèles reconnus dans le milieu comme Yolo, qui présente des résultats excellents dans toute sorte de tâche de détection, mais leur implémentation posait des difficultés que nous n'aurions pas su résoudre, et l'entraînement de réseau de cette taille n'aurait pas été possible sur nos machines au vu du temps qu'elles prenaient pour faire tourner des réseaux simples. Une approche basée sur ces modèles n'a donc pas été approfondie, mais aurait présenté assurément de meilleurs résultats.

8 Conclusion

Pour la méthode classique de détection de panneaux, bien que nous ayons réussi à construire un modèle de classification d'images très performant, la partie détection a été particulièrement difficile. En effet, les disparités entre les différents types de panneaux et la petite taille de notre ensemble de données d'apprentissage ont limité notre capacité à développer un modèle de détection efficace pour les panneaux.

Pour les méthodes par apprentissage profond, les mêmes constats sont fait. La partie détection pose réellement problème, la où la partie classification était plus simple. Le projet nous a cependant permis d'explorer plusieurs façons d'implémenter ces réseaux et d'utiliser leurs résultats.

Malgré cela, nous avons exploré toutes les techniques d'apprentissage classiques et de détection enseignées en cours. Ce projet nous a offert une occasion précieuse d'approfondir nos connaissances sur ce sujet, en particulier sur la complexité de la détection d'objets dans les images.