

Travail de Bachelor

2018

MESURE DE L'OCCUPATION DE PARKINGS À L'AIDE DE CAMÉRAS VIDÉO

Auteur
Rémi JACQUEMARD

Responsable
Juergen EHRENSBERGER



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Hes•so

Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Remerciements

Ce projet n'aurait pas pu être réalisé sans l'aide de quelques personnes, que je souhaitais particulièrement remercier.

En premier lieu, je tenais à remercier tout particulièrement Monsieur Juergen Ehrensberger, mon responsable de travail de Bachelor, pour l'aide, les conseils et le suivi effectué tout au long de ce projet.

Je souhaitais également remercier Monsieur Frédéric Gudit et Monsieur Eric Ladem, qui ont permis la bonne installation de la caméra et sa connexion au réseau local, et ce malgré les nombreux problèmes qui ont été rencontrés.

Il m'est important de remercier Monsieur Bastian Gardel, qui a mis en place la machine virtuelle locale à la HEIG-VD nécessaire à ce projet. Aussi, je remercie Monsieur Saman Handschin pour l'aide apportée lors du choix de la caméra, ainsi que pour la commande et la réception de celle-ci.

Je désire aussi remercier mes proches et ma famille pour leurs soutiens et l'aide qui a pu être apportée. Je souhaite notamment citer Monsieur Billy Jacquemard pour avoir résolu certains problèmes causés par la caméra, ainsi que Monsieur François Quellec pour l'aide apportée en ce qui concerne la suppression de l'arrière-plan d'une image.

Cahier des charges

Contexte

Avec les nouvelles techniques d'apprentissage automatique (*machine learning*) telles que le *deep learning*, il est devenu faisable de réaliser des applications d'analyse d'images au travers de l'entraînement à l'aide d'une collection d'images déjà catégorisées. Le but de ce projet est de d'utiliser ces méthodes afin de détecter des voitures sur un parking ouvert et ainsi d'en déduire l'occupation du parking. Le système à développer sera installé dans une première phase sur le toit du bâtiment de la HEIG-VD (route de Cheseaux) afin de mesurer l'occupation des différents parkings.

Cahier des charges

- Evaluation et choix d'un modèle de caméra
- Développement d'une infrastructure de captures d'images (caméras, réseau, serveurs cloud, stockage des images)
- Etablissement d'un corpus d'images prétraitées pour l'entraînement des algorithmes
- Prise en main des techniques d'apprentissage automatique
- Conception et développement de l'algorithme d'apprentissage automatique
- Tests et validation.

Table des matières

1	Résumé	8
2	Introduction	9
2.1	Contexte	9
2.2	Objectif	10
2.3	Etat de l'art	10
2.3.1	PKLot - A Robust Dataset for Parking Lot Classification	10
2.3.2	Segmentations de l'image	11
2.3.3	Détection des voitures	11
2.3.4	Conclusions	12
3	Bases technologiques	13
3.1	Traitemet d'images	13
3.1.1	Représentation d'une image	13
	Espace de couleur	13
3.1.2	Sous-échantillonnage	14
3.1.3	Détection de bords	15
3.2	Réseau de neurones à convolutions	16
3.2.1	Apprentissage automatique (<i>Machine Learning</i>)	16
3.2.2	Réseau de neurones	16
3.2.3	Réseau de neurones à convolutions	17
3.2.4	Détection d'objets	17
4	Conception	20
4.1	Architecture du système	20
4.1.1	Capture des images	20
	Architecture logique	20

Architecture physique	21
4.1.2 Distribution des informations à l'utilisateur	22
4.2 Choix technologiques	23
4.2.1 Caméra réseau	23
Contraintes	24
Critères	24
Caméras disponibles et évaluation	26
Choix final	33
4.2.2 Langage de développement	34
4.2.3 Traitement d'images	35
Librairies disponibles	35
Comparaison	35
Choix final	36
4.2.4 Réseau de neurones à convolutions	36
Tensorflow	36
Keras	36
Tensorflow Object Detection API	36
Darknet – Yolo	36
4.3 Corpus d'images	37
4.4 Traitement des images	38
4.4.1 Evaluation des différentes méthodes et choix	39
4.5 Conception des solutions	43
4.5.1 Suppression d'arrière-plan	43
4.5.2 Régression	45
4.5.3 Grille de détection	47
4.5.4 Limitations d'un développement complet de réseau de neurones	49
4.5.5 API de détection d'objets	50
Tensorflow	50
Yolo	52
4.6 Evaluation des modèles	52

5 Réalisation	53
5.1 Disponibilité des fichiers sources	53
5.2 Structure du projet	53
5.3 Capture d'images	54
5.3.1 Emplacement de la caméra	54
5.3.2 Configuration des périphériques	54
5.3.3 Requêtes et protocole	55
5.3.4 Agent	56
5.3.5 Monitoring	56
5.4 Traitement des images	57
5.5 Traitement du corpus <i>PKLot</i>	60
5.6 Mise en place et entraînements des modèles	60
5.6.1 Machine virtuelle	60
5.6.2 Format <i>VOC XML</i>	60
5.6.3 <i>Tensorflow Object Detection API</i>	61
5.6.4 <i>Yolo</i>	61
5.7 Utilisation du modèle final <i>Tensorflow</i>	62
6 Tests et validation	64
6.1 TensorFlow Object Detection API	64
6.2 Yolo v3	66
6.3 Choix final	66
6.4 Tests en conditions réels	67
7 Conclusions	68
7.1 Bilan du projet	68
7.1.1 Résultats obtenus	68
7.1.2 Réalisation	68
7.1.3 Améliorations possibles	69
7.2 Bilan personnel	70
A Résultats des traitements d'images	72

A.1	Sous-échantillonnages	72
A.2	Détection de bords, sous-échantillonnage	73
A.3	Détection de bords, sous-échantillonnage	75
B	Planification	77
C	Réalisation	79
Outils utilisés		80
Authentification		81
Bibliographie		84
Table des figures		85

1 Résumé

Pour les utilisateurs de parkings, il est souvent souhaité connaître le nombre de places libres disponibles, tout du moins s'il en reste. Aussi, être capable d'obtenir des statistiques d'utilisations des parkings peut être important pour leur gérant : par exemple, savoir distinguer les heures creuses des heures pleines permet d'optimiser les plages horaires des employés. C'est une problématique importante, qui a souvent été résolue par l'utilisation de divers capteurs.

Ce travail propose une solution se reposant sur un système de capture vidéo. Des images provenant d'une caméra, il est possible d'en retirer le taux d'occupation du parking à l'aide de diverses méthodes d'analyse d'images ou d'apprentissage automatique. Ce projet permet d'exposer avant tout des méthodes de *deep learning*, tel que la détection d'objets.

Afin de concrétiser les solutions explorées dans ce projet, une caméra a été installée sur le toit du site de Cheseaux la HEIG-VD. Celle-ci permet de capturer des images d'un des parkings qui y sont présents. L'évaluation des différentes caméras adéquates et l'installation de celle qui semble le plus adaptée est documentée dans ce travail.

Ce projet propose d'étudier plusieurs solutions qui peuvent répondre à la problématique de la mesure du taux d'occupation d'un parking. L'une de celle-ci se repose sur une suppression de l'arrière-plan d'une image. Cependant, il est souhaité que ce travail aborde avant tout le problème à l'aide de technologies d'apprentissages automatiques, et non pas de traitement d'images. Néanmoins, la solution pré-citée est brièvement explorée.

Des modèles de réseaux de neurones sont aussi présentés. Certains de ceux-ci sont définies à la main. D'autres utilisent ce qui a été pensé dans le domaine de la recherche de la détection d'objets, dans le but de pouvoir distinguer, dans une image, les voitures qui y sont présentes. Notamment, les modèles *Faster-RCNN* et *Yolo* sont présentés et testés.

2 Introduction

2.1 Contexte

Depuis des années, il est possible pour un utilisateur de connaître le nombre de places disponibles dans un parking. Celles-ci sont en effet souvent indiquées à leur entrée. Les différentes approches pour résoudre ce problème se reposent généralement sur une utilisation de capteurs de proximités qu'il est nécessaire de poser près de chacune des places de parc, ou encore à un comptage des voitures en entrée du parking à l'aide de barrières, notamment si un péage est présent.

Ce travail de Bachelor permet de proposer des solutions à la problématique de la détection du taux d'occupation d'un parking selon une approche qui diffère des usages originaux. En effet, l'installation de capteurs ou d'un système de ticket à l'entrée d'un parking semble souvent coûteux à mettre en place. Si le parking dont on souhaite installer un système de comptage de place libre est gratuit, les approches classiques ne semblent pas appropriées.

Au cours des dernières années, le domaine de l'apprentissage automatique (*Machine learning*) a connu de grandes avancées et s'est très largement démocratisé, grâce notamment à l'augmentation de la puissance de calcul. Par diverses méthodes de *deep learning* et à partir d'images préalablement annotées, il est possible d'entrainer des modèles afin d'effectuer de l'analyse d'images. En constante évolution, la recherche dans le domaine de la détection d'objets s'est grandement améliorée. Il est devenu possible, à l'aide de grandes bases de données annotées, de distinguer certains objets désignés d'une image.

Cette approche est celle qui est privilégiée dans ce projet. Des images de parking provenant de caméras les filmant peuvent être traitées et analysées algorithmiquement, dans le but d'en ressortir le nombre de places libres et le taux d'utilisation du parking.

L'utilisation de caméras semble apporter plusieurs avantages face aux approches classiques. En effet, leurs installations n'est que peu coûteuse en temps et en ressources. De plus, des caméras de sécurité sont souvent déjà présentes dans des parkings : seul l'installation d'un serveur permettant le traitement des images est nécessaire.

Deux approches différentes auraient pu être possibles afin de mesurer le taux d'occupation d'un parking à l'aide de caméra vidéo. Il aurait été possible de filmer l'entrée du parking, et de compter le nombre de voitures entrantes et sortantes. Cette approche aurait eu l'avantage qu'une seule caméra soit suffisante afin d'être en mesure de connaître le nombre de places disponibles dans le parking. Cependant, un problème de taille semble se poser : si une voiture qui entre ou sort n'est pas détectée, le nombre total de véhicules présents dans le parking est faussé, et le sera tant que ce nombre n'aura pas été manuellement recalibré. Ce projet aborde donc le problème différemment, où ce sont les emplacements de parkings qui sont filmés. Ainsi, un véhicule qui n'aurait pas pu être détecté ne fausse pas indéfiniment les résultats.

Ce travail permet aussi d'apporter des solutions aux problèmes des parkings "sauvages", soit ceux sans marquage au sol. Il sera possible de donner comme exemple les manifestations où, souvent, une pelouse est prévue pour stationner les véhicules. Dans ce cas, l'ajout de capteur n'est pas envisageable. Plus généralement, lorsque les parkings sont en plein air, il y a souvent un manque de marquage au sol, où les voitures peuvent aussi être stationnées le long de la route. Ce travail de Bachelor aborde ce

problème en cherchant non pas à compter le nombre de places de parking occupées, mais le nombre de voitures présentes sur le parking.

2.2 Objectif

Afin de répondre à la problématique décrite, une caméra sera installée sur le toit de la HEIG-VD du site de Cheseaux. Une première étape consiste donc en l'évaluation de différentes caméras disponibles dans le commerce et le choix du modèle convenant le mieux.

Dans un second temps, l'infrastructure de capture d'images sera mise en place. La caméra choisie devra donc être achetée et installée. Un serveur permettra de capturer les images à intervalles régulier. En ce qui concerne la capture des images, un soin tout particulier sera mis sur la protection de la sphère privées des utilisateurs du parking.

Il est important de noter que ce projet permet de poser les bases de la détection du taux d'occupation d'un parking. Par faute de temps et de moyens, une seule caméra sera utilisée, ne couvrant qu'une partie du parking du site de Cheseaux de la HEIG-VD. Il est évident que s'il est nécessaire d'ajouter plusieurs caméras dans le but de capturer des images de l'entier du parking, la détection du nombre de véhicules total n'est pas trivial : il est en effet nécessaire de pouvoir distinguer les voitures présentes sur plusieurs caméras à la fois.

Un *dataset* d'images annotées devra être mis en place pour l'entraînement des algorithmes de *machine learning*. Celui-ci peut être créé à partir des images de la caméra placée sur le toit de la HEIG-VD, ou un corpus d'images annotées peut être disponible en ligne.

La majeure partie de ce travail réside dans la recherche des meilleures méthodes disponibles pour répondre à la problématique de la détection du nombre de places disponibles d'un parking, en retirer le meilleur choix, le concevoir et le mettre en place. Bien évidemment, les modèles seront testés et validés.

2.3 Etat de l'art

La mesure du taux d'occupation d'un parking à l'aide de caméra vidéo est un sujet qui a déjà été abordé dans le domaine du *Machine Learning*. Ici est présenté quelques papiers traitant cette problématique.

2.3.1 PKLot - A Robust Dataset for Parking Lot Classification

PKLot est premièrement un large corpus d'images de parkings prises de 3 caméras différentes. Il a été créé dans le but de former un *dataset* robuste spécialisé pour la recherche dans le domaine des parkings. Les images ont été capturées sous diverses conditions météorologiques, et chaque place de parc a été annotée, les classifiant de libre ou occupée.¹

Il a été souhaité de le préciser ici, car il sera utilisé comme *dataset* pour ce projet (voir section 4.3 *Corpus d'images*). Aussi, quelques pistes de recherches sont proposées, reposant avant tout sur la

1. P. ALMEIDA et al. « PKLot – A robust dataset for parking lot classification ». In : *Expert Systems with Applications* 42 (2015), p. 4937-4949.

segmentation des images par place de parc. Il faut aussi noter que les voitures présentent sur des places non marquées n'ont pas été annotées.

2.3.2 Segmentations de l'image

Le papier *Deep Learning for Decentralized Parking Lot Occupancy Detection* présente un système de détection de places libres de parking, basé sur un réseau de neurones convolutif. Utilisant notamment le corpus *PKLot*, la détection se repose sur la segmentation de l'image par place de parc. Le réseau de neurones est entraîné sur chacune de ces parties, les classifiant en tant que place libre ou occupée. Cette approche apporte de bons résultats, où les taux de réussites peuvent dépasser les 99%.²

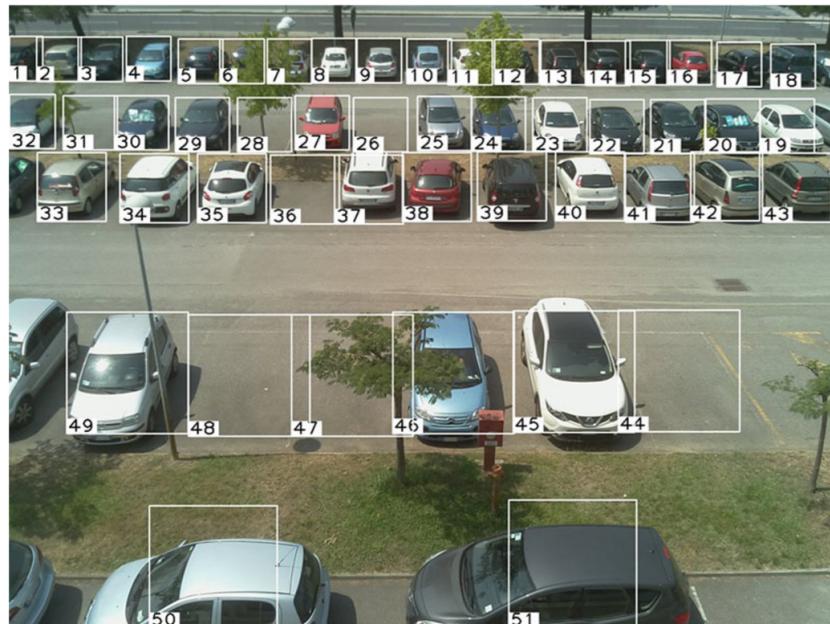


FIGURE 2.1 – Exemple d'images du corpus utilisé

De la même manière, le papier *Parking Stall Vacancy Indicator System Based on Deep Convolutional Neural Networks* permet d'obtenir des résultats similaires.³

2.3.3 Détection des voitures

Bo Li, Tianfu Wu et Song-Chun présentent dans leur papier une méthode de détection de voitures à partir d'images. Ceux-ci utilisent une méthode complexe basée sur des convolutions pour détecter certaines parties des voitures (phares, portes, etc.). Sur le corpus d'images *PKLot*, les résultats atteignent un taux de 55.2% de réussite.⁴

2. Giuseppe AMATO et al. « Deep Learning for Decentralized Parking Lot Occupancy Detection ». In : *Expert Syst. Appl.* 72.C (avr. 2017), p. 327-334. ISSN : 0957-4174. DOI : 10.1016/j.eswa.2016.10.055. URL : <https://doi.org/10.1016/j.eswa.2016.10.055>.

3. Sepehr VALIPOUR et al. « Parking Stall Vacancy Indicator System Based on Deep Convolutional Neural Networks ». In : *CoRR* abs/1606.09367 (2016). arXiv : 1606.09367. URL : <http://arxiv.org/abs/1606.09367>.

4. Tianfu WU, Bo LI et Song-Chun ZHU. « Learning And-Or Models to Represent Context and Occlusion for Car Detection and Viewpoint Estimation ». In : *CoRR* abs/1501.07359 (2015). arXiv : 1501.07359. URL : <http://arxiv.org/abs/1501.07359>.

2.3.4 Conclusions

En général, le problème de mesure du taux d'occupation d'un parking à l'aide de caméras vidéos est approchée à l'aide d'une segmentation des différentes places de parcs, ce qui est pertinent si les places disponibles sont toutes marquées au sol. Néanmoins, l'approche abordée dans ce travail diffère dans le but de répondre aux problèmes présentés en section *Contexte* tel que les parkings sauvages. Dans cette optique, on cherchera à détecter les voitures présentes, et non pas à classifier chaque place de parc comme étant libre ou non.

3 Bases technologiques

Ce chapitre permet de présenter quelques bases théoriques nécessaires à la bonne compréhension de ce travail.

3.1 Traitement d'images

3.1.1 Représentation d'une image

Une image est formée de pixel. La figure 3.1 présente une image d'une taille de 104x71 pixels, où ceux-ci peuvent donc être distingués les uns des autres.



FIGURE 3.1 – Image de faible définition dont les pixels sont apparents

FILL. *Parking automobiles véhicules*. 13 juin 2015. URL : <https://pixabay.com/fr/parking-automobiles-v%C3%A9hicules-825371/> (visité le 18/05/2018)

Chaque pixel d'une image couleur est généralement constitué de 3 valeurs¹ : une composante rouge, une verte et une bleue, qui permet de décrire la couleur associé à ce pixel. En traitement d'image, ces composantes RGB² sont souvent décrites comme des canaux³ différents.

Une image couleur peut donc être représentée en mémoire à l'aide d'un tableau à 3 dimensions. 2 normes différentes peuvent être utilisées :

Canal en premier (*channel first*) Le canal est désigné par la première dimension. Ainsi, pour une image couleur de 4x4 pixels, le tableau est de la forme 3x4x4.

Canal en dernier (*channel last*) Le canal est désigné par la dernière dimension. Pour une image couleur de 4x4 pixel, le tableau est de la forme 4x4x3.

Dans ce projet, il a été défini que le format *canal en dernier* sera utilisé en priorité.

Espace de couleur

En section précédente, le format RGB a été présenté. Il est cependant important de noter qu'il existe d'autres modèles permettant de représenter une couleur. Notamment, le modèle HSV (*Hue, Saturation,*

1. Parfois, une composante de transparence peut aussi être présente. Dans certains cas bien spécifiques, il peut même être possible d'y inclure des composantes telles que la profondeur (la distance entre la caméra et l'objet désigné par le pixel), ou encore la température (caméra thermique).

2. *RGB* : Red Green Blue pour rouge vert bleu

3. *channel*, mot anglais, est aussi utilisé.

0	150	150	150
255	150	150	150
0	150	150	150
150	0	0	0
150	0	0	0
150	0	255	255
150	255	0	0
150	255	0	0
150	255	0	0
150	255	255	255
150	255	255	255
150	255	255	255

FIGURE 3.2 – Représentation d'une image couleur 4x4 pixels et les composantes rouges, vertes, et bleues

Value, en français *TSV* pour *teinte, saturation, valeur*) a été utilisé dans ce travail.⁴

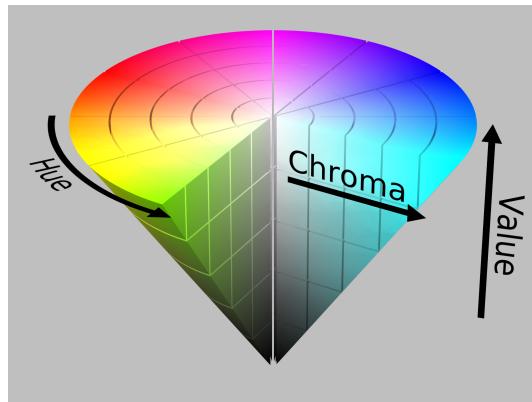


FIGURE 3.3 – Cône représentant l'espace HSV. Ici, la saturation est désignée par le terme *Chroma*
WIKIPEDIA. *HSL and HSV*. 28 juin 2018. URL : https://en.wikipedia.org/wiki/HSL_and_HSV (visité le 09/07/2018)

La teinte d'une couleur pouvant être obtenue grâce à l'aide d'un canal unique, cette représentation peut sembler idéale dans certains cas, permettant de mieux distinguer les couleurs entre elles. Notamment, cet espace de couleur a été utilisé lors des tests de détection de bords tel que décrit en section 4.4 *Traitement des images*.

3.1.2 Sous-échantillonnage

Une image d'un certain nombre de pixel en largeur et en hauteur peut être redimensionnée. En traitement d'images⁵, on parle de :

Sur-échantillonnage (*Oversampling*) La résolution de l'image est augmentée, le nombre de pixels qui y sont présents est donc supérieur.

Sous-échantillonnage (*Downsampling*) La résolution de l'image est réduite, le nombre de pixels qui y sont présents est donc diminué.

4. WIKIPEDIA. *HSL and HSV*. 28 juin 2018. URL : https://en.wikipedia.org/wiki/HSL_and_HSV (visité le 09/07/2018).

5. Le sur- et sous-échantillonnage sont en réalité utilisés pour toutes données décrites à l'aide d'échantillons ou une discréttisation, comme par exemple les fichiers audios.

Il est important de noter que la réduction de la résolution d'une image induit irrémédiablement à une perte d'information.

3.1.3 Détection de bords

Certains filtres permettent de mettre en valeur sur une images les bords des objets. Il sont souvent basés sur des matrices de convolutions. Celles-ci sont décrites plus amplement en section en section 3.2.3 *Réseau de neurones à convolutions*.



FIGURE 3.4 – Image de parking avant détection de bord

P. ALMEIDA et al. *Parking Lot Database - Laboratório Visão Robótica e Imagens*. URL : <https://web.inf.ufpr.br/vri/databases/parking-lot-database/> (visité le 18/05/2018)



FIGURE 3.5 – Image de parking après détection de bord

P. ALMEIDA et al. *Parking Lot Database - Laboratório Visão Robótica e Imagens*. URL : <https://web.inf.ufpr.br/vri/databases/parking-lot-database/> (visité le 18/05/2018)

Ces filtres sont généralement appliqués sur des images noires et blanches. Cependant, il est possible de séparer une image couleur par canal, et d'appliquer le filtre sur chacun de ceux-ci. C'est ce qui a été fait dans ce projet.

3.2 Réseau de neurones à convolutions

3.2.1 Apprentissage automatique (*Machine Learning*)

Le *Machine Learning* est un sous-domaine de l'intelligence artificielle. Comme son nom l'indique, il permet à une machine d'apprendre par divers moyens statistiques et automatiques.⁶ Ainsi, un algorithme permettant de réaliser une certaine tâche n'est pas directement défini par le développeur, mais "déduit" par apprentissage automatique.

Dans le cadre de ce TB, ce sont des algorithmes de type supervisé qui sont utilisés. L'apprentissage est effectué à l'aide de données labélisées. On distingue donc :

Les données Dans ce projet, il s'agit avant tout des images de parking.

Les labels Il s'agit de la sortie de l'algorithme qui est souhaité en fonction des données en entrée.

Dans l'absolu ici, il s'agit de connaître le taux d'occupation du parking pour une image de parking donnée.

3.2.2 Réseau de neurones

Dans ce projet, les algorithmes utilisés se basent sur des réseaux de neurones. Cette section permet d'en présenter le fonctionnement général et l'application de ceux-ci dans le cadre de ce TB.

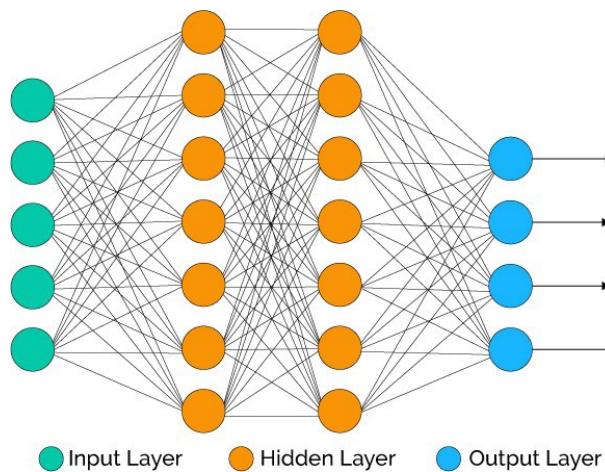


FIGURE 3.6 – Exemple de réseau de neurones

Donor McDONALD. *Machine learning fundamentals (II): Neural networks*. 21 déc. 2017. URL : <https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef> (visité le 17/07/2018)

La figure 3.6 présente une représentation minimale d'un réseau de neurone. On distinguera 3 couches, où chacune contient un certain nombre de neurones :

Couche d'entrée (*Input layer*) Chaque neurone de cette couche représente une partie des données d'entrées. Il est possible de faire un certain parallèle avec ce projet, où la couche d'entrée représente une image, et chaque neurone les pixels de celle-ci. Il faut cependant noter qu'avec un réseau de neurones à convolution (ce qui est utilisé dans ce projet), les choses sont légèrement différentes comme il sera possible de le voir en section 3.2.3.

6. WIKIPEDIA. *Machine Learning*. 29 juin 2018. URL : https://en.wikipedia.org/wiki/Machine_learning (visité le 17/07/2018).

Couche de sortie (*Output layer*) Chaque neurone de cette couche représente une sortie de l'algorithme qu'on souhaite obtenir. Bien que dans l'absolu, il est souhaité de connaître le nombre de voitures présentes dans un parking (un seul neurone de sortie serait suffisant), il sera vu en section 3.2.4 qu'il y en aura généralement plusieurs.

Couches cachées (*Hidden layers*) Elles permettent de mettre en relation les données d'entrées et les sorties voulues, ce dans le but, après entraînement, de "déduire" certaines caractéristiques des données d'entrées.

L'étape d'entraînement consiste à utiliser les données labélisées sur un réseau de neurones. Dans le cadre de ce projet, une image de parking est mise en entrée du réseau, ainsi qu'indirectement le nombre de voitures présentes dans l'image en sortie. Les paramètres du modèles sont modifiés itérativement, au fil des images présentées dans le *dataset*.

3.2.3 Réseau de neurones à convolutions

A un réseau de neurones conventionnel, il est possible d'y ajouter des couches de convolutions.

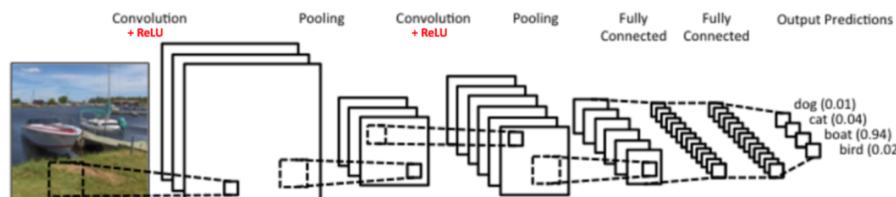


FIGURE 3.7 – Exemple de réseau de neurones à convolutions
Donor McDONALD. *An Intuitive Explanation of Convolutional Neural Networks*. 11 août 2016. URL : <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/> (visité le 17/07/2018)

Une couche de convolution consiste en un certain nombre de filtres, qui seront appliqués à une image. D'une image entrante dans cette couche, on y applique une convolution. En sortie, une nouvelle image, filtrée, est générée.

Pour résumer, une matrice de convolution décrit un filtre. Cette matrice est appliquée sur l'ensemble des pixels d'une image. Une fois ce filtre appliqué, chacun des pixels de cette image en est donc modifié. Une couche de convolutions transforme donc des images d'entrées en image de sorties filtrées.

Il semblait nécessaire et important de décrire rapidement le fonctionnement de ces matrices de convolutions afin de faire le parallèle avec la détection de bords qui a été décrites en section 3.1.3 *Détection de bords*. En effet, la détection de bord consiste en une matrice de convolution définie qui sera appliquée sur une image. En *Machine Learning*, les matrices de convolutions sont "déduites" au fur et à mesure des itérations. Ainsi, il est possible qu'un réseau de neurones à convolutions apprenne de lui-même à effectuer une détection de bord, si tel est nécessaire. C'est pourquoi, ce traitement peut accélérer l'apprentissage, mais ne semble pas dans l'absolu nécessaire.

3.2.4 Détection d'objets

Une approche possible afin de résoudre le problème présenté est de chercher à détecter les voitures présentes dans l'image. Une fois ces objets détectés, il est suffisant ensuite de compter le nombre de voitures prédites par l'algorithme.

Il faut noter qu'en général, ces algorithmes permettent de détecter plusieurs classes d'objets. Ainsi, ils peuvent parfois sembler excessivement complexes pour la tâche qui doit être effectuée dans le cadre de ce travail. Cependant, ces algorithmes sont ceux qui semblent fonctionner le mieux.

En général, ces algorithmes se basent sur des réseaux de neurones à convolutions tels que vu en section précédente. La première couche prend en entrée une image. La couche de sortie, quant à elle, dépend des architectures choisies. Il en existe beaucoup : ainsi, dans ce projet, le temps n'a permis de n'en tester que deux, et c'est ceux-ci qui sont présentés. Cependant, il est possible de noter que les architectures utilisées dans la détection d'objets sont souvent basées sur certains mêmes principes, où la présentation des deux méthodes suivantes permet de les résumer.

Faster R-CNN

L'architecture *Faster R-CNN*⁷ présentée en figure 3.8 consiste en 2 étapes. Dans un premier temps, un réseau de neurones à convolutions permet de rechercher les zones d'intérêts (*Regions of Interest*). Par la suite, chacune de ces zones est évaluée par un second réseau de neurones, afin de définir si la région appartient à certaines classes ou à aucune d'entre elles.

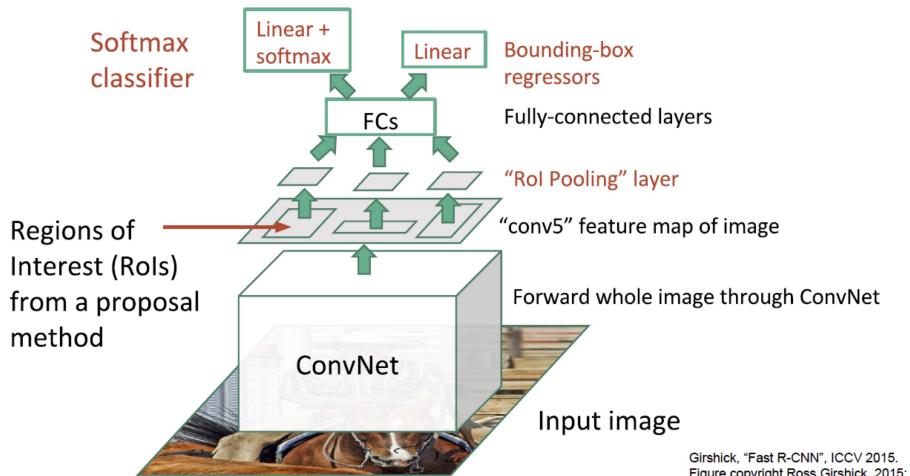


FIGURE 3.8 – Architecture *Faster R-CNN*

Shaoqing REN et al. « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ». In : *CoRR* abs/1506.01497 (2015). arXiv : 1506.01497. URL : <http://arxiv.org/abs/1506.01497>

Yolo

L'architecture présentée par *Yolo*⁸ consiste en un unique réseau de neurones à convolutions. Conceptuellement, par couche de convolutions successive, l'image est finalement réduite à une grille la subdivisant. Chaque cellule de cette grille indique s'il y a un objet détecté dans l'image originale à la position correspondante.⁹

7. Shaoqing REN et al. « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ». In : *CoRR* abs/1506.01497 (2015). arXiv : 1506.01497. URL : <http://arxiv.org/abs/1506.01497>.

8. Yolo (You Only Look Once). (Joseph REDMON et Ali FARHADI. « YOLOv3: An Incremental Improvement ». In : *arXiv* [2018])

9. Andrew NG, Kian KATANFOROOSH et Younes BENSOUDA MOURRI. *Object detection*. 2018. URL : <https://www.coursera.org/lecture/convolutional-neural-networks/object-localization-nEeJM> (visité le 27/06/2018).

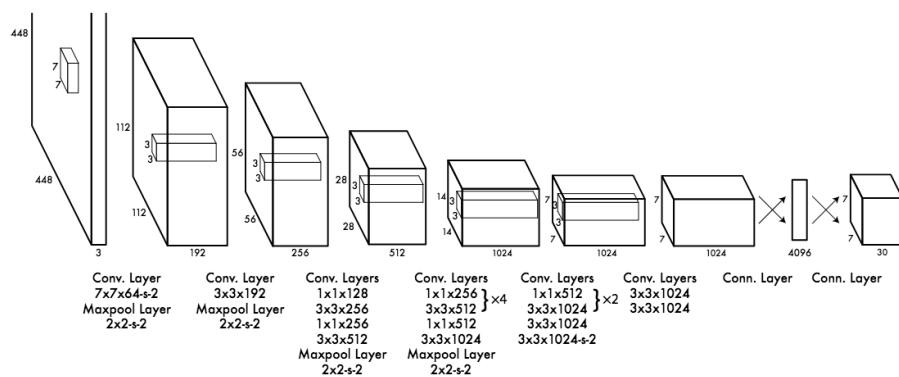


FIGURE 3.9 – Architecture Yolo

Joseph REDMON et Ali FARHADI. « YOLOv3: An Incremental Improvement ». In : *arXiv* (2018)

4 Conception

4.1 Architecture du système

4.1.1 Capture des images

Cette section présente l'architecture du système de capture d'image du parking, nécessaire à ce projet. Une caméra IP sera choisie, qui sera connectée au réseau de la HEIG-VD de la manière décrite ici. La capture d'image a été pensée afin de permettre la connexion de multiples caméras.

Il faudra remarquer que les choix effectués ici non seulement influencent la caméra qui sera choisie en section 4.2.1 *Caméra réseau*, mais aussi, ces choix dépendent de celle-ci. Ainsi, ces deux travaux (choix de l'architecture de capture d'image et choix de la caméra) ont été réalisés en parallèle.

Architecture logique

La caméra réseau sera connectée au réseau local de la HEIG-VD. Elle aura donc une adresse IP, qu'il sera possible d'utiliser afin de communiquer avec elle. Une machine virtuelle (abrégée *VM*, pour *Virtual Machine*) a été mise à disposition de ce projet. Située sur le réseau de l'école, elle permettra de récupérer ses images.

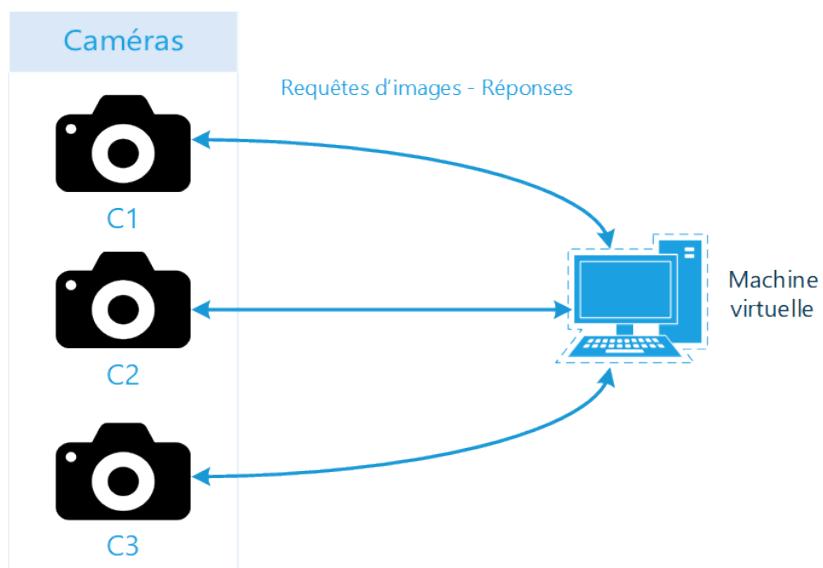


FIGURE 4.1 – Capture d'images de plusieurs caméras

La figure 4.1 présente un protocole de demande-réponse sur *HTTP* qui sera utilisé afin de récupérer des images à intervalles réguliers. Il est possible de la préciser ainsi :

1. La *VM* émet une requête demandant l'image du parking à une caméra.

2. La caméra répond à la VM avec ladite image.

Les requêtes et réponses précitées sont fortement dépendantes de la caméra choisie. Ainsi, les détails d'implémentation pourront être trouvés en section 5.3 *Capture d'images*.

Architecture physique

Positionnement de la caméra

La caméra doit pouvoir être capable de capturer des images de parking. Pour ce faire, une caméra sera installée sur le toit de la HEIG-VD.

Il faut remarquer que l'installation de périphériques sur la terrasse n'y est pas des plus aisée. De plus, l'édification d'une salle de conférence y a débuté en date du 26 février 2018¹, ne facilitant pas l'installation. Il en sera tenu compte lors du choix de la caméra. Plus d'informations concernant l'installation de celle-ci sont disponibles en section 5.3 *Capture d'images*.

Connexion de la caméra

Ici, deux points sont principalement étudiés : l'alimentation de la caméra, et sa connexion au réseau local de la HEIG-VD.

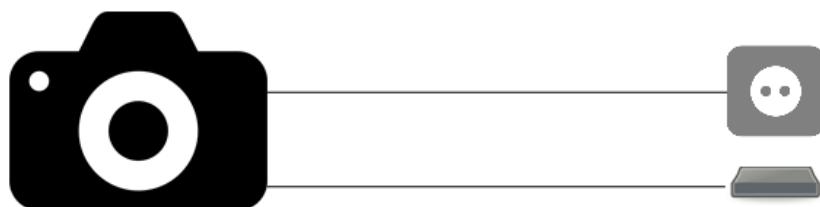


FIGURE 4.2 – Caméra alimentée et connectée au réseau par câble

La caméra pourrait être câblée au réseau électrique, ainsi qu'au réseau local. Il est nécessaire de tirer 2 câbles, à 2 endroits différents : l'un au réseau électrique, et l'autre à un switch ou une prise Ethernet présente dans le bâtiment.



FIGURE 4.3 – Caméra câblée au réseau et alimentée par PoE

Elle pourrait aussi être câblée uniquement au réseau local. En effet, il serait possible de profiter de la technologie *Power over Ethernet (PoE)* permettant de fournir en électricité un périphérique réseau via le câble Ethernet. Il faut cependant noter que le switch connecté à la caméra doit pouvoir fournir cette fonctionnalité.

1. Informations tirées d'un mail envoyé aux collaborateurs et étudiants de la HEIG-VD en date du 22 février 2018



FIGURE 4.4 – Caméra alimentée par câble et connectée au réseau en Wifi

La caméra pourrait être connectée au réseau électrique par câble. La connexion au réseau local pourrait être effectuée par Wifi. Il faut cependant noter que le Wifi doit être puissant et fiable : la caméra, installée à l'extérieur du bâtiment, pourrait ne pas capter suffisamment un signal provenant de l'intérieur, ceci dû aux larges murs en béton.

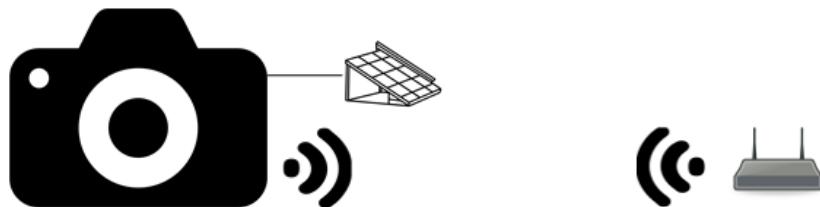


FIGURE 4.5 – Caméra auto-alimentée et connectée en Wifi

Une batterie, ou un panneau solaire, pourrait l'alimenter. Munie du Wifi, cette solution ne demanderait aucun câblage. On notera cependant que le prix d'un tel équipement pourrait être élevé.

Solution choisie

Comme indiqué au paragraphe *Positionnement de la caméra*, la construction d'une salle pourrait compromettre l'installation de câble. C'est pourquoi une caméra auto-suffisante sera préférée. Ainsi :

- La caméra devra pouvoir fournir une connexion puissante et fiable à un signal Wifi.
- Elle sera auto-alimentée. De préférence, un panneau solaire sera utilisé afin que des changements de batteries ne soient pas nécessaires.

4.1.2 Distribution des informations à l'utilisateur

Dans le but de pouvoir fournir le nombre de place libre à l'utilisateur, un simple *endpoint* a été développé à l'aide de *Flask*². Il faut noter que la distribution de ces informations n'est pas le point principal de ce projet, c'est pourquoi il est assez succinct.

D'autres systèmes auraient pu être mis en place, cependant, une telle API semblait être l'idéal. En effet, pour une application si simple, elle est rapidement développée. Aussi, une *REST API* est une technologie couramment utilisée et permet une certaine interopérabilité. Ainsi, une fois l'API exposée, d'autres applications l'utilisant peuvent être écrites, comme une interface web.

La détection des voitures à partir d'une image prend un certain temps, de l'ordre de la dizaine de secondes. Ainsi, plutôt que la prédiction soit effectuée à chaque requête sur l'*endpoint* exposé, ce qui

2. *Flask* est une librairie *Python* permettant de rapidement exposer une API *REST*.(*FLASK. Flask (A Python Micro-framework)*. URL : <http://flask.pocoo.org/> [visité le 21/07/2018])

bloque la réponse, il a été préféré que cette détection soit effectuée en arrière-plan. Pour ce faire, le script interroge la caméra à intervalle régulier à l'aide d'un agent (décris en section 5.3.4 *Agent*). La prédiction du nombre de voitures présentes est effectuées à l'arrivée d'une image, et est sauvegardée en mémoire.

Lorsqu'un utilisateur souhaite obtenir l'état actuel du parking, il interroge l'API *REST*. A la réception de la requête, le nombre de voitures présentes est lu en mémoire, et une réponse sous la forme d'un objet *JSON* contenant le nombre de places libres ainsi que d'autres informations est renvoyé à l'utilisateur. Le code *JSON* correspondant est présenté au listing 4.1.

Listing 4.1 – *JSON* reçu par l'utilisateur

```
1  {
2      "date" : "2018-07-25T12:38:38" ,
3      "free_place" : 17,
4      "num_cars" : 6,
5      "occupied_rate" : 0.2608695652173913
6 }
```

Afin de réduire les erreurs de prédiction, le nombre de voitures présentes sur le parking renvoyé à l'utilisateur est calculé à partir d'une moyenne des 4 dernières images prises par la caméra.

4.2 Choix technologiques

4.2.1 Caméra réseau

Une caméra réseau fournissant des images de qualités dans un prix raisonnable est nécessaire dans ce projet. Celle-ci sera utilisée principalement dans 2 buts, précisés ci-après.

Entrainement du modèle

Dans un premier temps, il s'agit d'entraîner le modèle à partir des images fournies par la caméra. Pour ce faire, il doit être possible de capturer et sauvegarder des photos à intervalles réguliers. Ces images seront ensuite annotées, puis utilisées pour entraîner le modèle de neurones à convolutions.

Utilisation du modèle

Une fois le modèle défini, le but de ce projet est de l'utiliser afin de détecter le nombre de voitures présentes sur le parking. Ainsi, dans un second temps, la caméra doit pouvoir fournir des images du parking "en direct", dans le but d'obtenir un état courant du parking. Il faut noter qu'ici, la notion "en direct" désigne un intervalle de capture "relativement court", soit de l'ordre de la minute. En effet, en tant qu'utilisateur, obtenir le taux d'occupation du parking à la minute près semble suffisant. Bien entendu, un intervalle plus court entre chaque prise, ou un flux vidéo, est un plus.

Il convient donc d'effectuer l'achat d'une caméra permettant le bon déroulement de ce projet. Pour ce faire, plusieurs caméras ont été évaluées. On rencontrera donc dans les prochaines sections les contraintes et critères nécessaires à une bonne évaluation. Elles seront notées, ce qui permettra d'en retirer celle qui convient le mieux à ce projet.

Il est important de noter qu'idéalement, des tests en condition réelle des différentes caméras auraient dû être effectués. Cependant, dans le cadre de ce travail de Bachelor, le budget mis à disposition ne peut le permettre. C'est pourquoi une seule caméra sera choisie sur la base des critères définis.

Contraintes

Comme vu en section 4.1.1 *Architecture physique*, installer une caméra sur le toit de la HEIG-VD a pour conséquence que certaines contraintes, tant d'ordre techniques qu'organisationnelles, doivent être respectées. Celles-ci sont explicitées dans cette section.

Capture d'images

La caméra devra au minimum permettre la capture d'images à intervalles réguliers d'une façon ou d'une autre. L'intervalle minimum de capture est de l'ordre de la minute, afin de pouvoir obtenir l'état courant du parking filmé. La capture de vidéos n'est pas nécessaire. On notera que si celle-ci ne fournit malheureusement que la fonctionnalité de capture vidéo, il serait tout de même possible d'en extraire des images utilisables.

Usage extérieur

La caméra doit être prévue pour un usage extérieur, avec une protection contre les intempéries (pluie, neige, froid, chaud, etc.)

Caméra auto-alimentée

Comme vu en section 4.1.1, la caméra doit être auto-alimentée et non câblée. Ainsi, une combinaison de batteries et de panneaux solaires semble être idéal. On peut noter que l'utilisation de batteries uniquement est envisageable dans le cadre d'un prototypage, mais ce uniquement si le remplacement de celles-ci doit être effectué à intervalles relativement éloignés, de l'ordre de la semaine.

Connexion réseau

En lien avec le paragraphe précédent, la caméra doit nécessairement avoir une interface Wifi permettant de se connecter au réseau de l'école. Une connexion par Ethernet n'est pas envisageable.

Résolution d'image

Dans le but d'obtenir des images de qualités suffisantes, la résolution de l'image capturée devra nécessairement être au minimum de 1280x720 pixels.

Critères

Plusieurs critères ont été définis. Chacune des caméras retenues sera notée selon ceux-ci. On les trouvera ci-après, où leur pondération sont indiqués entre crochets ([]). Une pondération élevée signifie une plus grande importance de ce critère.

Qualité d'image [1]

La caméra doit avoir une bonne qualité d'image. On jugera :

- La résolution de l'image. Celle-ci doit être d'au minimum 1280x720 pixels.
- La qualité de l'image (si possible).

Fonctionnalités réseaux [2]

La caméra doit pouvoir fournir des images via le réseau et il doit être possible d'en récupérer à intervalles réguliers. Pour ce faire, on pensera à des protocoles comme les suivants :

ONVIF (Open Network Video Interface Forum) Standard industriel ouvert permettant de contrôler, configurer et communiquer avec des caméras de sécurité IP. Permet notamment la lecture de flux vidéo en temps réel et la capture de photos³

RTSP (Real Time Streaming Protocol) Développé par RealNetworks, Netscape et Columbia University, protocole de communication permettant de lire en temps réel un flux vidéo.⁴

Requêtes HTTP Il pourrait être possible de capturer et de récupérer à la demande une photo à l'aide de requêtes HTTP.

Flux sur HTTP Permet la lecture de flux vidéo sur HTTP en s'appuyant sur des formats tel que MPEG-4.

Interface Web HTTP Permet la configuration de la caméra IP via un serveur Web qu'elle expose.

FTP La caméra peut exposer un serveur FTP contenant les photos capturées. Elle pourrait aussi téléverser des photos capturées sur un serveur FTP distant.

Il est important de préciser qu'on évaluera avant toute sur la facilité d'utilisation des différents protocoles.

Fonctionnement auto-alimenté [4]

La caméra doit être auto-alimentée et non câblée. Comme critères, on pensera donc à :

- La durée de vie de la caméra en fonctionnement auto-alimenté. Idéalement, la caméra ne devrait nécessiter aucune intervention humaine.
- La portée du Wifi. La puissance du signal doit être assez forte afin de pouvoir capter les bornes Wifi de l'école situées à l'intérieur depuis le toit de la HEIG-VD. Il est important de noter que ce critère peut être difficilement évalué avant l'achat de la caméra.

Capture de photo [2]

La caméra doit être capable de fournir un moyen pour capturer des photos à intervalles réguliers, de l'ordre de la minute. Pour ce faire, la caméra peut fournir un système de capture de photos automatique, ce qui est un plus. On évaluera donc les moyens fournis par la caméra permettant ces captures.

Angle de vue [2]

L'angle de vue de la caméra doit être suffisamment grand afin d'obtenir une image du parking dans son ensemble. Il peut être suffisant à partir de 40°, de par la hauteur à laquelle elle sera placée. Cependant, un angle de 90° semble plus satisfaisant. Ces angles ont été définis à l'aide des informations fournies par videosurveillance-boutique.fr.⁵

3. WIKIPEDIA. *ONVIF*. 18 fév. 2018. URL : <https://en.wikipedia.org/wiki/ONVIF> (visité le 07/03/2018).

4. WIKIPEDIA. *Real Time Streaming Protocol*. 27 jan. 2018. URL : https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol (visité le 07/03/2018).

5. VIDEOSURVEILLANCE-BOUTIQUE.FR. *Comment bien choisir sa caméra de surveillance*. URL : <http://www.videosurveillance-boutique.fr/support/comment-bien-choisir-sa-camera-de-surveillance-483.html> (visité le 12/03/2018).

Vision de nuit [1]

Idéalement, la caméra devrait pouvoir fournir une vision de nuit afin de pouvoir capturer des images de parking par toute heure. On pensera notamment aux matinées et aux soirées d'hiver, où les véhicules arrivent et partent alors que la luminosité est encore très faible. Cependant, dans le cadre de ce TB, cette fonctionnalité n'est pas strictement nécessaire.

Facilité d'installation [1]

La facilité d'installation de la caméra sera prise en compte. On pensera aux dimensions de celle-ci, à son poids, au nombre de ses composants (par exemple, est-il nécessaire d'installer une batterie en plus de la caméra, ou est-elle incorporée à celle-ci ?), ou encore aux accessoires fournis pour son installation.

Prix [4]

Bien évidemment, le prix doit entrer en ligne de compte. 100.- CHF sera indiqué comme prix maximum afin d'obtenir la meilleure note. A partir de 500.- CHF, on évaluera ce prix comme étant mauvais.

Caméras disponibles et évaluation

Bien que la surveillance n'est pas un but de ce projet, les caméras de sécurité semblent être les plus adaptées. En effet, elles fournissent généralement des fonctionnalités de capture de photos, de connexion réseau, de vision de nuit, possèdent un angle de vue suffisant et sont souvent résistantes à un environnement extérieur. De plus, elles sont spécialement conçues pour un usage qui s'apparente à celui de ce TB, et sont donc adaptées à la prise d'images de parking.

L'achat distinct d'une caméra, de panneaux solaires et de batteries est envisageable. Néanmoins, dans un premier temps, seuls des kits complets (caméra, batteries, panneaux solaires) ont été analysés. En effet, l'installation et le choix des différents composants nécessaires à un système solaire complet semble difficile lorsqu'on est pas du domaine, et des problèmes d'interopérabilité pourraient survenir. Ceci reste cependant une solution viable si aucun kit ne correspond aux contraintes définies.

De part les contraintes très spécifiques précisées jusqu'ici, le nombre de caméras les satisfaisant toutes est faible. On trouvera ci-après les kits complets solaires auto-suffisants qui ont été évalués. Il faut aussi noter que seules les caméras les plus appropriées sont indiquées ici.

Electrosun – Caméra-surveillance-solaire

Electrosun est une entreprise de domotique solaire française. Entre autres, un kit solaire de caméra complet est mis en vente.⁶

Bien qu'elle satisfasse la plupart des contraintes, elle n'a pas été retenue pour l'évaluation. En effet, la résolution des images capturées n'est que de 640x480 pixels, ce qui n'est pas suffisant.

Reolink – Argus 2

L'entreprise *Reolink* développe des caméras 100% sans fil, avec batteries rechargeables et panneaux solaires. Elle est auto-suffisante.⁷

Elle permet de capturer des photos FullHD (1920x1080 pixels), possède une fonction de vision de nuit ou encore, l'utilisation en extérieur est possible. Cependant, cette caméra n'a elle non plus pas pu être

6. ELECTROSUN. *Caméra IP solaire autonome*. URL : <http://electrosun.fr/Cam%C3%A9ra-surveillance-solaire> (visité le 18/03/2018).

7. REOLINK. *Argus 2 - Wire-Free Camera*. 2018. URL : <https://reolink.com/product/argus-2> (visité le 13/03/2018).



FIGURE 4.6 – **Electrosun** Caméra de surveillance solaire

ELECTROSUN. *Caméra IP solaire autonome.* URL : <http://electrosun.fr/Cam%C3%A9ra-surveillance-solaire> (visité le 18/03/2018)



FIGURE 4.7 – **Reolink** Argus 2

REOLINK. *Argus 2 - Wire-Free Camera.* 2018. URL : <https://reolink.com/product/argus-2> (visité le 13/03/2018)

prise en considération. En effet, elle a été pensée pour un usage privé, et bien que la caméra puisse être connectée à un réseau Wifi, l'accès à celle-ci n'est possible qu'à l'aide d'une application smartphone (tel que décrit dans les spécifications disponibles sur le site officiel <https://reolink.com/product/argus-2>⁸). Récupérer des photographies à intervalles réguliers semble donc difficilement réalisable.

Wanscam – HW0029-3

Wanscam est une entreprise chinoise spécialisée dans la production de caméra réseau. Plusieurs versions de son modèle *HW0029* sont disponibles. Ici est présenté le modèle *HW0029-3*.⁹

Cette caméra satisfait toutes les contraintes définies en section 4.2.1. On trouvera en table 4.1 ses principales caractéristiques.¹⁰

8. REOLINK, cf. note 7.

9. WANSCAM. *HW0029-3 - Outdoor IP Camera.* URL : <http://wanscam.com/productshow-14-65-1.html> (visité le 11/03/2018).

10. Tirées des spécifications officielles (WANSCAM, *HW0029-3 - Outdoor IP Camera*, cf. note 9) et d'un test indépendant (Les GEEKS. [TEST] Présentation De La Caméra IP SOLAIRE HD 720p WANSCAM HW0029. 17 déc. 2016. URL : <https://lesbonstuyauxgeeks.fr/test-présentation-de-la-camera-ip-solaire-hd-720p-wanscam-hw0029/> [visité le 11/03/2018]).

FIGURE 4.8 – **Wanscam HW0029-3**

WANSCAM. *HW0029-3 - Outdoor IP Camera.* URL : <http://wanscam.com/productshow-14-65-1.html> (visité le 11/03/2018)

TABLE 4.1 – Caractéristiques de la caméra **Wanscam HW0029-3**

Caractéristique	Valeurs
Image	1280 x 720, 25fps, compression H.264
Angle de vue	40°
Vision nocturne	Visibilité jusqu'à 15 mètres
Réseau	RJ45, Wifi 802.11 b/g/n
	2 batteries de 12A et panneau solaire.
Auto-alimentation	Sans soleil : 48h d'utilisation. Faible température (<26°) des rayons de soleil : 7 jours d'utilisations. Grande température (>33°) des rayons de soleil : fonctionnement continu
Stockage	Carte MicroSD de 16Gb incluse, support jusqu'à 128Gb
Détection de mouvement	Oui
Accès aux images	Flux HTTP, ONVIF, RTSP, requêtes HTTP
Dimensions et poids	370x290x110 mm, 2.7 kg

On notera que le panneau solaire est fixé sur la caméra. Ainsi, dans le but de garder une exposition au soleil suffisante, l'angle de la caméra ne doit pas être trop élevé. Il y a donc certaines contraintes à l'installation de celle-ci.

Concernant la capture d'image, la caméra fournit des *endpoints HTTP* (dont une liste peut être trouvée sur le site tutoriels.domotique-store.fr¹¹) permettant de récupérer l'image actuelle. Il est donc facile d'imaginer un simple agent qui, à intervalle régulier, demandera à la caméra une image, et la sauvegardera en local.

Les caméras **Wanscam** ne sont pas des plus faciles à trouver dans le commerce. Elles sont principalement disponibles sur aliexpress.com. Le modèle *HW0029-3* peut être trouvé à partir de 170CHF (en date du 09.03.2018, aliexpress.com).

11. Damien ABAD - Domotique STORE. *API des CAMÉRAS IP Wanscam Onvif.* 29 avr. 2016. URL : <http://tutoriels.domotique-store.fr/content/52/293/fr/api-des-cameras-ip-wanscam-onvif-hw-nouvelle-generation.html> (visité le 11/03/2018).

La grille d'évaluation 4.2 a pu être remplie à l'aide de ces spécifications.

TABLE 4.2 – Evaluation de la caméra **Wanscam HW0029-3**

Critère	Note (de 1 à 5)	Remarques
Qualité d'image	3 [1]	La qualité d'image semble suffisante selon les tests effectués par lesbonstuyauxgeeks.fr . ¹² La résolution est de 1280x720 pixels, ce qui est suffisant.
Fonctionnalités réseaux	5 [2]	La caméra offre notamment un endpoint HTTP sur lequel récupérer les images.
Fonctionnement auto-alimenté	4 [4]	La caméra est totalement auto-suffisante s'il y a assez de soleil. On notera cependant le panneau solaire fixe qui peut nuire à l'exposition du soleil. Elle supporte le Wifi 802.11n : à première vue, le signal semble suffisant.
Capture de photo	5 [2]	Il est facile de récupérer des images sur cette caméra. Elle offre en plus un système de capture de photos à intervalles réguliers automatique. Elle offre aussi la possibilité de récupérer un flux vidéo.
Angle de vue	1 [2]	Un angle de vue de 40° semble tout juste suffisant.
Vision de nuit	2 [1]	Elle fournit une vision nocturne à 15m. Il est cependant possible d'ajouter des LED infrarouges supplémentaires afin d'obtenir une meilleure vision de nuit. ¹³
Facilité d'installation	3 [1]	La caméra mesure jusqu'à 37 cm, ce qui nuit à sa facilité d'installation. De plus, le panneau solaire est fixe, ce qui contraint la position et l'angle de la caméra.
Prix	4 [4]	A partir d'environ 170.- CHF
Note finale : 3.6/5		

Wanscam – HW0029-5

Le modèle *HW0029-5* de **Wanscam** est la nouvelle version du modèle *HW0029-3* décrit précédemment. Elle possède la plupart des mêmes caractéristiques, mais propose une meilleure résolution (1920x1080), des batteries à plus grande capacité, un meilleur champ de vision, ou encore, une vision de nuit accrue jusqu'à 100m. Son prix est cependant plus élevé que la caméra *HW0029-3*. Les caractéristiques indiquées dans ce rapport proviennent du site officiel wanscam.com.¹⁴

14. WANSCAM. *HW0029-5 - Outdoor IP Camera 1080P*. URL : <http://wanscam.com/productshow-16-94-1.html> (visité le 12/03/2018).

FIGURE 4.9 – **Wanscam HW0029-5**

WANSCAM. *HW0029-5 - Outdoor IP Camera 1080P.* URL : <http://wanscam.com/productshow-16-94-1.html> (visité le 12/03/2018)

TABLE 4.3 – Caractéristiques de la caméra **Wanscam HW0029-5**

Caractéristique	Valeurs
Image	1920 x 1080, 25-30fps, compression H.264
Angle de vue	80°
Vision nocturne	Visibilité jusqu'à 100 mètres
Réseau	RJ45 100Mbps, Wifi 802.11 b/g/n
Auto-alimentation	2 batteries de 12A et panneau solaire. Sans soleil : 54h d'utilisation.
Stockage	Carte MicroSD de 16Gb incluse, support jusqu'à 128Gb
Détection de mouvement	Oui
Accès aux images	Flux HTTP, ONVIF, RTSP, requêtes HTTP
Dimensions et poids	370x286x960 mm, 4.3 kg

Il faudra noter que cette caméra est plus lourde, bien que plus petite, que le précédent modèle.

De ces caractéristiques, il a été possible d'évaluer cette caméra dans la grille d'évaluation 4.4

TABLE 4.4 – Evaluation de la caméra **Wanscam HW0029-5**

Critère	Note (de 1 à 5)	Remarques
Qualité d'image	4 [1]	La qualité de l'image est meilleure que le modèle précédent, avec une résolution de 1920x1080 pixel.
Fonctionnalités réseaux	5 [2]	La caméra offre notamment un endpoint HTTP sur lequel récupérer les images.
Fonctionnement auto-alimenté	4 [4]	De la même manière que le modèle précédent, la caméra est totalement auto-suffisante s'il y a assez de soleil. Sans soleil, elle est auto-suffisante jusqu'à 54h. Elle supporte la norme Wifi 802.11n
Capture de photo	5 [2]	Il est facile de récupérer des images sur cette caméra. De plus, elle offre un système de capture de photos à intervalles réguliers automatiques. Elle offre aussi la possibilité de récupérer un flux vidéo.
Angle de vue	4 [2]	Un angle de vue de 80° semble être approprié pour ce projet. Un angle supérieur n'aurait pas été superflu.
Vision de nuit	5 [1]	Elle fournit une vision nocturne à 100m.
Facilité d'installation	3 [1]	Tout comme le modèle HW0029-3, la caméra n'est pas des plus petites, mesurant jusqu'à 37 cm. Le panneau solaire est fixé à la caméra.
Prix	3 [4]	A partir d'environ 230.- CHF
Note finale : 4.0/5		

Remarque

Le modèle **Wanscam HW0029-4** (ainsi que sa version améliorée **HW0029-6**) n'a pas été pris en compte. En effet, il fourni en plus un système de puce 4G afin de pouvoir connecter la caméra à un réseau mobile. En conséquence, le prix est trop élevé, dépassant les 400.- CHF.

Visortech – Caméra Solaire IP

Cette caméra satisfait toutes les contraintes définies dans la section 4.2.1 *Contraintes*. Ses caractéristiques principales sont décrites à la table 4.5. Il a été possible de remarquer que la caméra décrite ici est disponible sous différents noms, comme *IdealSmartCam*.

On regrettera le peu d'informations disponibles concernant les spécifications de cette caméra. Par exemple, il est difficile de connaître les méthodes disponibles afin de pouvoir récupérer des images depuis le réseau local, tant et si bien que cela soit possible. On notera que le vendeur a été contacté, sans réponse de sa part.



FIGURE 4.10 – Caméra Visortech

PEARL.CH. Caméra HD-IP. URL : <https://fr.pearl.ch/article/NX4377/camera-hd-ip-avec-batterie-rechargeable-et-panneau-solaire-ipc-790-solar> (visité le 09/03/2018)

TABLE 4.5 – Caractéristiques de la caméra Visortech

Caractéristique	Valeurs
Image	1280 x 720, compression H.264. Prise de vue en continu possible.
Angle de vue	90°
Vision nocturne	Visibilité jusqu'à 5 mètres
Réseau	Wifi 802.11 b/g/n
Auto-alimentation	Panneau solaire intégré de 0.8 Watt, jusqu'à 5 jours sans soleil. Auto-suffisante.
Détection de mouvement	Oui
Dimensions et poids	161x155x108 mm, 0.8 kg

Malgré le manque d'information, cette caméra a tout de même été évaluée. Il sera de ce fait possible de savoir si elle peut être un choix viable, ou non. La grille d'évaluation correspondante peut être trouvée en table 4.6

TABLE 4.6 – Evaluation de la caméra *Visortech*

Critère	Note (de 1 à 5)	Remarques
Qualité d'image	3 [1]	Sa résolution est de 1280x720 pixel.
Fonctionnalités réseaux	2 [2]	Accès via smartphone possible, peu d'informations concernant les protocoles réseaux disponibles.
Fonctionnement auto-alimenté	3 [4]	Devrait être auto-suffisante. Cependant, informations peu claires si prises d'images en continu. Accepte la norme Wifi 802.11n.
Capture de photo	3 [2]	Permet l'accès au flux vidéo en direct. Peu d'informations sur les protocoles disponibles.
Angle de vue	5 [2]	Angle de vue de 90°. Meilleur champ de vision parmi les caméras évaluées.
Vision de nuit	2 [1]	Elle fournit une vision nocturne à 5m.
Facilité d'installation	3 [1]	La caméra est relativement petite, et ne pèse que 850g. On notera l'absence de pied : une fixation à un mur semble nécessaire.
Prix	3 [4]	A partir d'environ 220.- CHF
Note finale : 3.1/5		

Choix final

La caméra *Visortech* arrive dernière du classement, avec une note de 3.1. Ceci est avant tout du au manque d'informations sur les protocoles utilisables afin de récupérer via le réseau des images. Elle n'a donc pas été retenue.

TABLE 4.7 – Caméras - Synthèse des résultats

	Caméra	Note
1	Wanscam HW0029-5	4.0
2	Wanscam HW0029-3	3.6
3	Visortech	3.1

Les deux versions de la caméra *HW0029* de **Wanscam** montrent des résultats proches. Elles sont avant tout distinguables par leur qualité d'image, leur vision de nuit, leur durée de fonctionnement sur batterie et leur angle de vue, où toutes ces caractéristiques sont améliorées sur le nouveau modèle (*HW0029-5*). Ainsi, seul le rapport qualité/prix peut faire pencher la balance d'un côté ou d'un autre. Avec un peu de recherche, la version *HW0029-5* a été trouvée à 187.50€ sur [aliexpress.com](https://fr.aliexpress.com/item/Wanscam-HW0029-3-720P-solar-wifi-IP-camera-build-in-16G-TF-cards-support-128G-TF/32795681508.html)¹⁵ (frais de livraison inclus), soit 225.- CHF (1.17CHF/€ en date du 13 mars 2018,¹⁶ arrondi au dixième

15. MAGICVISION. *Aliexpress.com – Wanscam HW0029-5 Extérieure Étanche*. URL : <https://fr.aliexpress.com/item/Wanscam-HW0029-3-720P-solar-wifi-IP-camera-build-in-16G-TF-cards-support-128G-TF/32795681508.html> (visité le 13/03/2018).

16. ZONEBOURSE.COM. *EURO / SWISS FRANC (EUR/CHF)*. URL : <https://www.zonebourse.com/EURO-SWISS-FRANC-EUR-C-4594/graphiques-cours/> (visité le 12/05/2018).

supérieur, soit 1.20 CHF/€).

Au prix indiqué plus haut, et vis-à-vis du classement présenté en table 4.7, la caméra **Wanscam HW0029-5** a été choisie pour la capture d'image que demande ce projet.

4.2.2 Langage de développement

Afin de choisir le langage de programmation qui sera majoritairement utilisé dans ce projet, il est nécessaire de définir quel en sera son utilisation. On résumera donc ses objectifs principaux ainsi :

- Le langage sera utilisé afin de capturer des images sur HTTP.
- Le langage sera fortement utilisé pour du traitement d'image.
- Le langage sera fortement utilisé pour du *Machine Learning*, plus spécifiquement des réseaux de neurones à convolutions.

De ce fait, se tourner vers un langage s'approchant du *script* (qu'on pourra par exemple opposer au langage *Java*) semble avoir plusieurs avantages :

- Le langage sera beaucoup utilisé afin de traiter des informations. Ces traitements seront relativement courts, mais multiples. Un langage de script semble suffisant.
- Les langages de scripts sont légers, aisément maintenables et évolutifs. Il est cependant important de noter qu'ils le sont seulement si les programmes développés restent courts et bien construits.
- Les scripts ressemblent souvent à de la programmation procédurale, ce qui peut en faciliter son écriture. Puisqu'avant tout, des traitements seront effectués, un style de programmation procédurale semble idéal.

Dans le cadre de ce TB, du *Machine Learning* sera effectué. Il convient donc aussi de s'informer sur les langages de développement les plus populaires utilisés dans ce contexte. On en tirera une liste, non-exhaustive. Il faut noter que cette liste a été définie par ce qui a pu être vu par l'expérience, et non par un réel classement :

R Environnement gratuit de statistiques et graphiques.¹⁷

MATLAB Développé par *MathWorks*, combine un environnement *desktop* pour de l'analyse et un langage de programmation spécialisés.¹⁸

Python Langage de programmation permettant un développement rapide et efficace.¹⁹

Le choix du langage dépend aussi fortement des librairies qui sont disponibles et utilisées. Dans le cadre du *Machine Learning* et du traitement d'images, elles sont très souvent disponibles en *Python*.

Choix du langage

De part ce qui a été précisé précédemment, *Python* semble être le langage idéal. Il est possible de noter qu'il permet aussi une certaine conception orientée objet.

Limitation

Une des principales limitations de ce langage et qu'il est interprété. Ainsi, les performances peuvent être plus faibles qu'un langage compilé²⁰. Cependant, et comme on le verra plus tard, les librairies

17. R-PROJECT. *The R Project for Statistical Computing*. URL : <https://www.r-project.org/> (visité le 14/06/2018).

18. MATLAB. *MATLAB - MathWorks*. URL : <https://www.mathworks.com/products/matlab.html> (visité le 14/06/2018).

19. PYTHON.ORG. *Welcome to Python.org*. URL : <https://www.python.org/> (visité le 14/06/2018).

20. On notera cependant qu'il est possible, si on le souhaite, de compiler du *Python*. Là n'est cependant pas son but premier

fournies sont souvent des *wrappers* interfaçant du code compilé. Ainsi, les traitements lourds sont souvent efficaces, où *Python* peut ne servir qu'à connecter des composants et définir des configurations.

4.2.3 Traitement d'images

Dans ce projet, une des premières étapes consiste à traiter les images provenant de la caméra afin qu'elles puissent être utilisées convenablement dans l'algorithme de *Machine Learning*. On trouvera ci-après les différents traitements qui doivent pouvoir être réalisés :

Downsampling L'image reçue de la caméra sera sous-échantillonnée.

Edge detection Les bords pouvant être détectés dans l'image devront pouvoir être mis en valeur.

Librairies disponibles

Plusieurs librairies *Python* sont disponibles. Ici en sont décrites deux.

OpenCV

*OpenCV*²¹ (*Open Source Computer Vision Library*) est une librairie Open Source permettant le traitement d'image. Elle peut être utilisée en C++, Java, et Python. Elle est écrite en C/C++ et optimisée pour une utilisation sur un CPU multi-core. Elle permet notamment de sous-échantillonner une image et de détecter des bords, mais possède aussi beaucoup de fonctionnalités liées au *Machine Learning*.

Scikit-image

Scikit-image,²² abrégé *skimage*, est une collection gratuite et Open Source d'algorithmes pour le traitement d'images en python. Elle permet aisément de sous-échantillonner des images, ainsi que de détecter des bords à l'aide de filtres prédéfinis (convolutions).

Comparaison

Les deux librairies pré-citées sont ressemblantes sur bien des points. Une image, dans les deux cas, est représentée sous la forme d'un tableau *numpy*²³ à 3 dimensions. Les syntaxes des deux librairies sont aussi extrêmement proches où, parfois même, les noms de méthodes et arguments présents se trouvent être strictement les mêmes.

On pourra cependant les différencier par deux points principaux, en relation avec ce projet :

- La librairie *skimage* est plus facile d'accès. Ces méthodes sont mieux définies et le traitement d'une image est plus aisé qu'avec *OpenCV*. Elle offre, en plus, des fonctionnalités d'aide au développement pour le traitement d'images tri-signaux (valeurs *RGB*) et leurs conversions (par exemple, en valeurs *HSV* telles que décrites en section 3.1.1 *Espace de couleur*)
- La librairie *OpenCV* est plus complète. Elle possède en plus des fonctionnalités liées au *Machine Learning*, comme de la détection d'objet dans une image.

21. OpenCV TEAM. *OpenCV library*. URL : <https://opencv.org/> (visité le 14/06/2018).

22. SCIKIT-IMAGE.ORG. *Scikit-image: Image processing in Python*. URL : scikit-image.org (visité le 14/06/2018).

23. Numpy est une librairie scientifique permettant d'utiliser en *Python* des tableaux multidimensionnels de manière efficace.(NUMPY.ORG. *NumPy*. URL : <http://www.numpy.org/> [visité le 14/06/2018])

Choix final

Il a donc été choisi d'utiliser au possible la librairie *scikit-image*, qui est plus facile d'accès.

Néanmoins, il est important de noter que le fait qu'une image soit représentée de la même manière dans les deux librairies permet une certaine interopérabilité entre celles-ci. *Skimage* souffre de certaines limitations lorsqu'on la compare à *OpenCV*, mais il est facile de les contourner : lorsque c'est nécessaire, *OpenCV* pourra être utilisé en parallèle.

4.2.4 Réseau de neurones à convolutions

Cette sous-section présente les diverses librairies et API d'apprentissage automatique utilisées dans ce projet. Plusieurs modèles ont été testés, et il a été nécessaire d'utiliser plusieurs librairies différentes.

Tensorflow

*TensorFlow*²⁴ est une librairie open-source de *machine learning* développée par *Google*. Elle permet de définir, entraîner, tester et utiliser des réseaux de neurones, notamment à convolutions. Elle peut être utilisée en *Python*.²⁵

Keras

Keras est une API de réseau de neurones haut niveau écrite en *Python*, permettant d'en simplifier grandement le développement. Elle permet le choix entre plusieurs *backend*, dont *Tensorflow*.²⁶

Tensorflow Object Detection API

Cette API open-source a été développée par *Google* et des collaborateurs. Elle permet de construire, entraîner et déployer des systèmes de détection d'objets sur *TensorFlow*.²⁷ Des modèles pré-entraînés basés sur des architectures reconnues sont notamment disponibles²⁸.

Darknet – Yolo

Il existe une implémentation de l'architecture *Yolo* (qui a été décrite en section 3.2.4 *Détection d'objets*). Elle utilise *Darknet* comme *framework* open-source de réseau de neurones.²⁹

24. TENSORFLOW. *TensorFlow*. URL : <https://www.tensorflow.org/> (visité le 19/07/2018).

25. WIKIPEDIA. *TensorFlow*. 28 juin 2018. URL : <https://en.wikipedia.org/wiki/TensorFlow> (visité le 22/07/2018).

26. KERAS. *Keras: The Python Deep Learning library*. URL : <https://keras.io/> (visité le 22/07/2018).

27. KERAS. *Tensorflow Object Detection API*. URL : https://github.com/tensorflow/models/tree/master/research/object_detection (visité le 18/06/2018).

28. Les modèles pré-entraînés sont disponibles à l'adresse https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

29. PJREDDIE. *YOLO: Real-Time Object Detection*. URL : <https://pjreddie.com/darknet/yolo/> (visité le 26/06/2018).

4.3 Corpus d'images

Afin de pouvoir entraîner un *réseau de neurones*, un *dataset* d'images annotées est nécessaire.

Deux approches sont possibles :

- Annotation manuelle d'images de parking. Pour ce faire, les images capturées par la caméra de la HEIG-VD peuvent être annotées.
- Utilisation de corpus d'images de parking disponible en ligne.

La deuxième approche a été préférée, car cela permet une plus grande liberté. En effet, les images provenant du parking de la HEIG-VD sont pré-traitées pour des raisons d'anonymisations des données. L'utilisation d'un corpus d'images disponibles permet de pouvoir traiter les images après-coup, et ce seulement si c'est ce qui est souhaité. Aussi, l'utilisation d'un *dataset* déjà annoté permet d'obtenir un grand nombre de photos qui ont été capturées sur plusieurs mois, où la variance des luminosités et couleurs est grande. De plus, le fait qu'un corpus soit déjà annoté permet un gain de temps certain. Il faudra cependant noter qu'en conséquence, toute l'étude effectuée dans ce rapport n'est pas spécialisée pour le parking de la HEIG-VD. Dans tous les cas, un modèle viable sur des images d'autres parkings peut être considéré comme *proof-of-concept*.

Deux *datasets* ont avant tout été utilisés. Ceux-ci sont décrits ci-après.

Cars

Il a été souhaité de tester un entraînement à l'aide du corpus d'images *Cars*,³⁰ de l'université de Stanford. Celui-ci contient plus de 16'000 images de 196 modèles de voitures différents, sous différents angles. Il est annoté à l'aide de *bounding box*, ainsi que le modèle de voiture associés.



FIGURE 4.11 – Images d'exemple provenant du dataset *Cars*

PKLot

PKLot est un *dataset* de plus de 12'000 images provenant de 3 caméras différentes. Les photos ont été capturées sous plusieurs conditions météorologiques (beau temps, nuageux, pluie). Il est annoté à l'aide de *bounding box* précisant les places de parc. Pour chacune d'entre elle, il est indiqué si la place est libre ou non.

30. Jonathan KRAUSE et al. « 3D Object Representations for Fine-Grained Categorization ». In : *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.



FIGURE 4.12 – Image d'exemple provenant du dataset *PKLot*

4.4 Traitement des images

Cette section présente le traitement des images qui est effectué sur les captures prises par la caméra de la HEIG-VD dans le but de pouvoir créer un *dataset* qui pourrait être utilisé dans un réseau de neurones. Il faut noter que celui-ci ne sera pas obligatoirement utilisé pour l'entraînement du modèle final.

Anonymisation

Dans le cadre de ce TB, un point essentiel est l'anonymisation des données. En effet, la sphère privée des utilisateurs du parking du site de Cheseaux se doit d'être respectée. Un effort tout particulier sera donc pris afin d'anonymiser au mieux ces images. Notamment, les images capturées de la caméra sont traitées dès leurs arrivées, et ce même avant leurs enregistrement sur disque persistant.

Aussi, dans ce rapport, les images de parking présentées proviennent du dataset *PKLot*³¹. De ce fait, aucune image du parking de la HEIG-VD où des personnes et des modèles de voitures pourraient être distingués n'y sera présent. Il est important de remarquer que les résultats obtenus dans cette section en utilisant ce dataset sont équivalents à ceux obtenus à partir de la caméra du site de Cheseaux. Ainsi, les paramètres définis ici sont aussi valables afin de traiter les images provenant du parking de la HEIG-VD. L'image présentée en figure 4.13 est celle qui sera utilisée afin d'effectuer les différents tests présent dans cette section.

Sous-échantillonnage (Downsampling)

Dans le but d'obtenir des images légères qu'un réseau de neurones peut traiter aisément et qu'aucune personne ou voiture ne puisse être reconnue, l'image est sous-échantillonnée. L'image peut être réduite à des tailles différentes : c'est ce qui sera testé dans cette section.

Détection de bord

Il est possible de traiter les images en les filtrant à l'aide d'un filtre. Dans cette section, les matrices de convolutions suivantes, qui semblent donner les meilleurs résultats, ont été testées :

31. *PKLot* (disponible à l'adresse <https://web.inf.ufpr.br/vri/databases/parking-lot-database/>) est un grand dataset de plus de 12'000 images provenant de 3 parkings, libre à l'utilisation.(ALMEIDA et al., « PKLot – A robust dataset for parking lot classification », cf. note 1)



FIGURE 4.13 – Image d'exemple provenant du dataset *PKLot* qui sera utilisée afin de définir les divers paramètres

- Sobel filter
- Scharr filter

Il est important de noter que ces filtres ne peuvent être appliqués qu'à une image en valeurs de gris. Cependant, comme il a été défini en section 3.2.3 *Réseau de neurones à convolutions*, il est possible de séparer une image à 3 canaux en 3 images distinctes, d'effectuer le filtre sur chacun de ceux-ci, et de les recombiner en une image 3 canaux par la suite.

Aussi, l'image a été convertie dans plusieurs représentations différentes, car en effet, les résultats obtenus peuvent en être modifiés. Les suivantes ont été testées :

Valeur de gris L'image peut être convertie en valeur de gris. En sortie, une image à un seul canal est produite.

RGB L'image est représentée sous la forme de valeurs RGB de chacun des pixels. Elle est donc à 3 canaux, et les filtres seront appliqués sur chacun de ceux-ci.

HSV L'image est convertie dans l'espace de couleur HSV. De la même manière que la représentation RGB, les 3 canaux résultants sont filtrés séparément.

Il est à noter que l'application d'un filtre de détection de bords a un grand rôle dans l'anonymisation des données, où des visages ou des modèles de voitures ne peuvent pratiquement pas être distingués dans les images en résultant.

4.4.1 Evaluation des différentes méthodes et choix

Dans cette section sont donc comparés différents filtres, résolutions et représentations d'images.

Il faudra noter que le traitement peut être effectué de 2 manières différentes : soit l'image est premièrement sous-échantillonnée, soit les bords sont détectés. La méthodologie suivie afin de tester au mieux les différentes possibilités est la suivante :

1. Les différentes possibilités de sous-échantillonnages et de détection de bords ont premièrement été explorés séparément.
2. Le traitement "détection de bords en premier, suivi d'un sous-échantillonnage" a été testé.

3. De la même manière, le traitement "sous-échantillonnage en premier, suivi d'une détection de bords" a lui aussi été exploré.
4. Finalement, les deux traitements précisés en 2 et 3 ont été comparés.

Sous-échantillonnage

Dans un premier temps, on comparera différentes résolutions d'images. Celles-ci sont les suivantes :

- 1280 x 720 pixels
- 1120 x 630 pixels
- 960 x 540 pixels
- 800 x 450 pixels
- 640 x 360 pixels
- 480 x 270 pixels
- 320 x 180 pixels
- 160 x 90 pixels

Il est possible de trouver en annexe A.1 les résultats complets du rééchantillonnage en fonction des diverses résolutions.



FIGURE 4.14 – Image originale rééchantillonnée à 320 x 180

Il semble que la résolution de 320x180 pixel, présentée en figure 4.14 soit suffisante à bien distinguer les voitures. Cependant, on notera que celle-ci pourrait ne pas l'être après application du filtre de détection de contours.

Détection de bords

Les filtres *Sobel* et *Scharr*, ainsi que les représentations RGB, HSV et en valeurs de gris ont été testés. La figure 4.15 en montre les résultats si ces filtres sont appliqués sur l'images originales. Sur celle-ci, on distinguera les 2 filtres par les colonnes, et les 3 représentations par les lignes.

(a) Filtre *Sobel* - RGB(b) Filtre *Scharr* - RGB(c) Filtre *Sobel* - HSV(d) Filtre *Scharr* - HSV(e) Filtre *Sobel* - Valeurs de gris(f) Filtre *Scharr* - Valeurs de gris

FIGURE 4.15 – L'image originale avec différentes détections de bords

Les résultats sont assez similaires. Evidemment, la représentation en valeurs de gris fait perdre toutes notions de couleur. Entre les deux autres représentations, on peut remarquer que l'espace HSV permet de plus facilement distinguer les différentes couleurs. Celles-ci ont leurs importances dans ce projet : en effet, une voiture rouge est facilement distinguable de l'herbe verte.

Concernant les deux filtres *Sobel* et *Scharr*, il n'y a que très peu de différences. Cependant, il semble que la matrice de convolution de *Scharr* offre une détection de bords légèrement meilleure.

Ainsi, un filtre *Scharr* appliqué sur l'espace de couleur *HSV* semble le plus indiqué. Il faut noter que ceci peut dépendre du sous-échantillonnage. Concernant l'espace de couleur *HSV*, si celui-ci ne convient pas pour le *Machine Learning*, il sera toujours possible de convertir les images en valeurs de gris.

Ordre de traitements

Il a ensuite été décidé de l'ordre d'application des deux traitements. Les résultats suivants sont disponibles en annexe :

A.2 Une détection de bords est effectuée en premier, suivi d'un sous-échantillonnage.

A.3 Un sous-échantillonnage est effectué en premier, suivi d'une détection de bords.

Comparaison des deux méthodes de traitement

Il est possible de comparer les deux méthodes de traitement définies précédemment. La figure 4.16 présente leur résultat côte-à-côte.



(a) Ici, les bords de l'image ont été détectés, puis celle-ci a été réduite à une taille de 480 x 270 pixels



(b) Ici, l'image a premièrement été réduite à une taille de 480 x 270 pixels, puis les bords ont été détectés

FIGURE 4.16 – Comparaison des deux méthodes de traitements

La différence principale entre ces deux images réside dans les bords détectés, où la méthode *b* permet de les mettre plus en valeur. Les traits sont plus foncés et épais. Certaines voitures sont moins distinguables dans la méthode *a*, se fondant dans le fond.

Il sera donc possible de résumer le traitement qui sera effectué sur l'image de la manière suivante :

1. L'image sera réduite à une taille de 480 x 270 pixels

2. Un filtre *Scharr* sera appliqué sur chacun des canaux de l'espace HSV de cette image.

Ce traitement a été appliquée à une image provenant de la caméra placée sur le toit du bâtiment du site de Cheseaux de la HEIG-VD. La figure 4.17 en présente les résultats.



FIGURE 4.17 – Image traitée, capturée depuis la caméra placée sur le toit de la HEIG-VD

Comme on le voit, les voitures présentes restent distinguables, sans pour autant nuire à la sphère privée des utilisateurs. Les modèles de voiture ne sont par exemple pas distinguable.

Il convient cependant de préciser qu'un tel pré-traitement induit une certaine perte d'information et pourrait ne pas convenir à l'entraînement d'un algorithme d'apprentissage automatique, où certaines voitures pourraient n'être pas assez visibles. Aussi, il est possible que la mise en valeurs des bords ne soit pas adaptée à la détection des voitures. Cependant, il n'était pas envisageable d'enregistrer des images non ou peu traitées du parking de la HEIG-VD afin de protéger la vie privée des utilisateurs.

4.5 Conception des solutions

Cette section permet de présenter les solutions qui ont été pensées pour résoudre le problème de détection du taux d'utilisation du parking.

4.5.1 Suppression d'arrière-plan

Ici est présenté une solution de traitement d'images qui peut permettre de résoudre la problématique.

Celle-ci se base sur un système de suppression d'arrière-plan. Ici, une méthode simple est présentée, où une image est soustraite à l'autre. La figure 4.18 présente les entrées et sorties de l'algorithme.

Dans un premier temps, le choix d'une image d'arrière-plan est nécessaire. Celle-ci ne doit contenir aucun véhicule (Image 4.18a).



(a) Arrière plan - Image de parking vide



(b) Image de parking avec voitures



(c) Différence entre l'image avec voitures et l'arrière plan

FIGURE 4.18 – Suppression d'arrière plan

P. ALMEIDA et al. *Parking Lot Database - Laboratório Visão Robótica e Imagens*. URL : <https://web.inf.ufpr.br/vri/databases/parking-lot-database/> (visité le 18/05/2018)

De l'image dont on souhaite connaître le nombre de véhicules présents (4.18b), on soustrait l'arrière-plan, pixel par pixel. Lorsque les pixels en résultant sont en-dessous d'un certain seuil défini, ils sont considérés arrière-plan. Sinon, il s'agit de l'avant-plan. L'image 4.18c en présente un résultat. Il est possible de remarquer qu'elle est bruitée, notamment vers les arbres en bas de la photo. Il paraît cependant aisément de la débruiter en supprimant les petits groupes de pixels formant l'avant-plan qui sont isolés, par exemple avec une matrice de convolution.

Afin d'obtenir une bonne approximation du taux d'occupation, il est possible, à l'aide d'un certain nombre d'images annotées, de statistiquement définir une fonction $f(x)$, prenant en entrée le nombre

de pixel de l'avant-plan, et en sortie le taux d'utilisation.

Une approche plus simple, mais moins fiable, consiste aussi à simplement traiter une image du parking plein, de compter le nombre de pixels m faisant partie de l'avant-plan, et de considérer cette valeur m comme étant le maximum de pixel possible désignant des voitures. La fonction linéaire $t(x)$ présentée ci-dessous permet de calculer le taux d'utilisation du parking en fonction du nombre de pixel formant l'avant-plan.

$$t(x) = cx \quad (4.1)$$

$$t(m) = 1 = cm \implies c = \frac{1}{m} \quad (4.2)$$

$$t(x) = \frac{1}{m}x \quad (4.3)$$

L'utilisation d'une telle méthode de traitement d'image pourrait être appliquée. Cependant, il semble que cette approche peut poser quelques problèmes. Parmi ceux-ci, on peut citer :

- La luminosité, les couleurs et les ombres changent au fil d'une journée et des saisons. Pour utiliser cette approche, il semble nécessaire d'acquérir un grand nombre d'images du même parking vide, sous différentes luminosités. Cela peut paraître parfois difficile : en effet, un parking n'est que rarement vide en pleine journée.
- L'angle de la caméra doit être fixe et ne doit pas être modifié au fil des captures d'images. Il semble cependant que les mouvements de la caméra doivent pouvoir être compensés à l'aide de points de repères identifiables algorithmiquement placé sur certains objets de l'image.

La méthode décrite ici paraît viable. Cependant, la réalisation de celle-ci semble devoir prendre beaucoup de temps, de part la résolution des problèmes précédemment cités. Aussi, un algorithme plus généralisé, permettant d'être porté sur d'autres parkings aisément, aurait été souhaité. Ici, la capture d'images d'arrière-plan avec différentes luminosités doit être réitérée avec chaque parking.

4.5.2 Régression

La première approche qui a été pensée est d'aborder le problème comme une simple régression.

Un tel problème consiste en une fonction dont la sortie est quantitative, et non qualitative.³² Dans notre cas, cela signifie qu'en sortie de l'algorithme, on cherche le nombre de voitures présentes sur l'image présentée en entrée.

Pour ce faire, un réseau de neurones à convolutions, comme décrit en section 3.2.3 *Réseau de neurones à convolutions* peut être utilisé.

On trouvera en figure 4.19 un modèle qui peut être utilisé, et qui a été testée. Le schéma se lit de haut en bas, de la couche d'entrée à la couche de sortie. Ce modèle a été défini à l'aide de la librairie *Keras*

³². WIKIPEDIA. *Régression (statistique)*. 24 oct. 2017. URL : [https://fr.wikipedia.org/wiki/R%C3%A9gression_\(statistiques\)](https://fr.wikipedia.org/wiki/R%C3%A9gression_(statistiques)) (visité le 19/07/2018).

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 268, 478, 32)	864
batch_normalization_1 (Batch Normalization)	(None, 268, 478, 32)	128
activation_1 (Activation)	(None, 268, 478, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 133, 238, 32)	0
conv2d_2 (Conv2D)	(None, 131, 236, 64)	18432
batch_normalization_2 (Batch Normalization)	(None, 131, 236, 64)	256
activation_2 (Activation)	(None, 131, 236, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 130, 235, 64)	0
dropout_1 (Dropout)	(None, 130, 235, 64)	0
flatten_1 (Flatten)	(None, 1955200)	0
dense_1 (Dense)	(None, 200)	391040200
activation_3 (Activation)	(None, 200)	0
dropout_2 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 1)	201
activation_4 (Activation)	(None, 1)	0
<hr/>		
Total params:	391,060,081	
Trainable params:	391,059,889	
Non-trainable params:	192	

FIGURE 4.19 – Proposition de modèle de réseau de neurones à convolutions pour résoudre le problème de régression

En entrée, une image de parking est passée, qui sont traitée par une première couche de convolutions. En sortie, un simple neurone permet d'indiquer le nombre total de voitures présentes sur l'image.

Il est aussi possible d'y remarquer une succession de couches telles que les suivantes :

Batch normalization Permet de normaliser les données, dans le but d'accélérer l'apprentissage.

Max pooling Permet de réduire la taille des images en entrée.³³

Activation La couche des fonctions d'activations³⁴.

33. C'est une méthode classique utilisée dans les réseaux de neurones convolutifs. En effet, une telle couche de *max pooling* est généralement présente après une couche de convolutions. (WIKIPEDIA. Réseau neuronal convolutif. 5 juin 2018. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif [visité le 19/07/2018])

34. Une fonction d'activation est généralement associée à chaque neurone dans les réseaux neuronaux (WIKIPEDIA. Réseau de neurones artificiels. 13 juil. 2018. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels [visité le 17/07/2018])

Flatten D'images en entrée (tenseur³⁵ à 2 dimensions), cette couche permet de les aplatisir en un tenseur à 1 dimension, nécessaire en entrée d'un réseau de neurones classique.

Dropout Permet de mieux généraliser. Certaines relations entre les neurones de deux couches sont simplement supprimées.

Dense Ajoute une couche de neurones classiques.

Il faut noter que les données de sortie ont été normalisées dans le but d'accélérer l'entraînement. Ainsi, la sortie, plutôt que d'indiquer le nombre de véhicule, indique le taux d'utilisation du parking, entre 0 et 1. Cette valeur peut être facilement calculée lorsqu'on connaît le nombre maximal de voitures qui peuvent y être présentes. La fonction suivante permet de le calculer, où v est le nombre de voitures présentes, et c la capacité du parking :

$$f(v) = \frac{v}{c}$$

Les diverses fonctions d'activations ne sont pas présentées sur la figure 4.19. Deux différentes ont été utilisées :

Relu (Rectified Linear Unit) Simple fonction linéaire qui transforme les valeurs négatives en entrée en 0 en sortie. Très souvent utilisée dans les réseaux de neurones à convolutions. Toutes les fonctions d'activations présentes, mis à part celle de sortie, sont une *Relu*.

Sigmoid Fonction d'activation qui transforme les valeurs d'entrée en une valeur de sortie entre 0 et 1. Elle est utilisée en sortie, ce qui permet d'obtenir le taux d'occupation du parking aisément.

Il faut noter que le modèle présenté n'en est qu'un parmi plusieurs autres qui ont été testés. Cependant, les résultats finaux n'ont pas été concluants, et il ne sera pas plus amplement exploré dans ce rapport. Il est cependant possible d'émettre quelques hypothèses pouvant expliquer ces disfonctionnements :

- Le problème à résoudre peut ne pas être si simple. Un réseau de neurones plus complexe et plus profond, en multipliant le nombre de couches, doit sans doute être nécessaires.
- Le label *taux d'utilisation du parking* ne donne au réseau de neurones aucune indication sur ce qu'il doit prendre en compte et distinguer dans les images. Indiquer où sont les voitures peut sembler plus efficaces.
- Le nombre d'itérations effectuées lors des tests peut être trop faible. En effet, lorsqu'on observe ce qui est fait dans le domaine de la détection d'objet, plusieurs dizaines d'itérations sont en générales effectuées.

L'utilisation d'une fenêtre glissante a aussi été pensée. Cette méthode consiste en le découpage d'une image en de multiples sous-images, en utilisant différentes tailles de fenêtres. Ensuite, on cherche à classifier chacune de ces sous-régions en fonction de si une voiture y est présente ou non. Cependant, cette manière de faire n'est que peu indiquée, car trop coûteuse en calculs et en ressources.

4.5.3 Grille de détection

Le modèle précédent décrit en section 4.5.2 n'a pas permis d'obtenir de résultats concluants. Il a notamment été précisé qu'une indication concernant la position des voitures semblait pouvoir per-

35. Un tenseur peut être comparé à un tableau à n dimension. Un tenseur à 2 dimension est tout simplement une matrice.

mettre d'améliorer les résultats. De part cette hypothèse, il a été pensé qu'il est sans doute possible de chercher à obtenir en sortie de l'algorithme la position des voitures plutôt que leur nombre.

Pour ce faire, il a été pensé de subdiviser l'image d'entrée en une grille. Cette idée est inspirée par ce qui est fait dans l'algorithme *Yolo*, décrit en section 3.2.4. La figure 4.20 présente une subdivision de 5×4 sur une image de parking. En chaque cellule de la grille, on cherchera à savoir si une voiture y est présente.

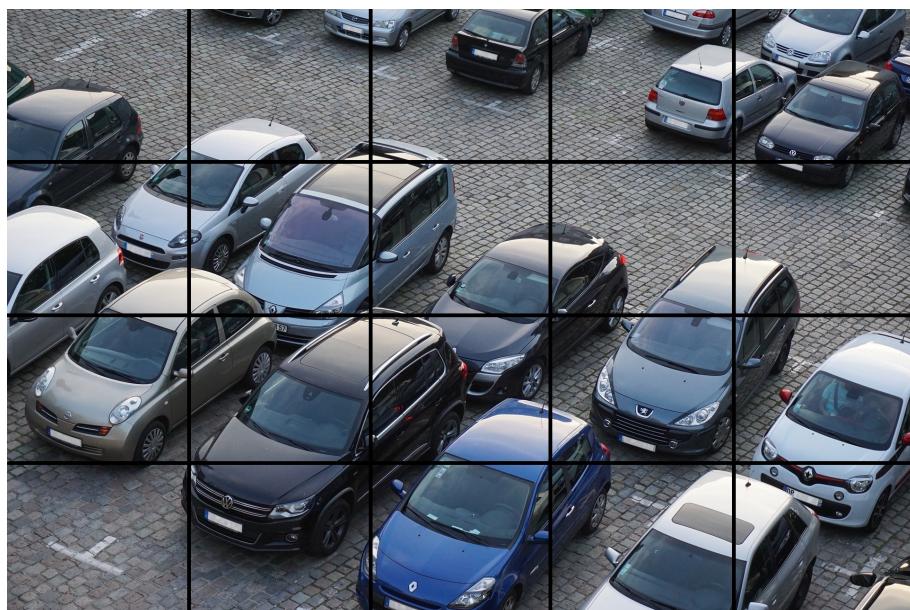


FIGURE 4.20 – Image de parking subdivisée selon une grille

FILL. *Parking automobiles véhicules*. 13 juin 2015. URL : <https://pixabay.com/fr/parking-automobiles-v%C3%A9hicules-825371/> (visité le 18/05/2018)

Il faut noter qu'une cellule contient une voiture seulement si le centre de l'objet y est présent. Ainsi, chaque voiture est détectée une seule fois. Aussi, la grille présentée en figure 4.20 a une faible dimension, où il est possible qu'une cellule contienne deux voitures. En réalité, la grille doit avoir des dimensions supérieures à 5×4 .

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 718, 1278, 32)	864
batch_normalization_1 (Batch Normalization)	(None, 718, 1278, 32)	128
activation_1 (Activation)	(None, 718, 1278, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 358, 638, 32)	0
conv2d_2 (Conv2D)	(None, 356, 636, 64)	18432
batch_normalization_2 (Batch Normalization)	(None, 356, 636, 64)	256
activation_2 (Activation)	(None, 356, 636, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 355, 635, 64)	0
zero_padding2d_1 (ZeroPadding2D)	(None, 374, 638, 64)	0
conv2d_3 (Conv2D)	(None, 22, 22, 1)	31553
activation_3 (Activation)	(None, 22, 22, 1)	0
<hr/>		
Total params: 51,233		
Trainable params: 51,041		
Non-trainable params: 192		

FIGURE 4.21 – Proposition de modèle de réseau de neurones à convolutions, avec une grille en sortie

La figure 4.21 présente un modèle de réseau de neurones. Celui-ci reprends principalement le modèle développé précédemment en section 4.5.2 *Régression*. On remarque cependant que la sortie diffère : en lieu et place d'un unique neurone de sortie, un tensor 2D y est présent. Chaque cellule de ce tableau peut prendre des valeurs entre 0 et 1 (fonction d'activation *sigmoïde*), où 1 indique la présence d'une voiture, et 0 son absence.

Les labels en sortie n'indiquent donc plus le nombre de véhicules présents, mais s'il y a un en chaque cellule de l'image. C'est donc un tableau à deux dimensions.

Lors des quelques tests effectués avec des modèles de tailles diverses, il semble que le principe décrit peut être fonctionnel. Cependant, il n'a pas été largement exploré et a été mis de côté, de part les grandes complexités nécessaires et les temps d'entrainements longs, tels que décrits en section suivante.

4.5.4 Limitations d'un développement complet de réseau de neurones

Les deux modèles précités consistent en un développement d'un réseau de neurones à l'aide de la librairie *Keras*. Cependant, plusieurs limitations de cette façon d'approcher le problème ont été remarquées, ce qui a eu pour conséquence la mise en pause de leur développement et de l'explorations des différents modèles et paramètres. Elles sont décrites ici.

Dans un premier temps, il faut noter que le développement de réseau de neurones adéquat pour la recherche d'objet dans une image prend un certain temps. En effet, il a été remarqué lors de la

découverte des *API* de détection d’objets que les réseaux neuronaux sont d’une grande complexité, où le nombre de couche dépasse souvent la quinzaine. Afin de définir un modèle soit même, il est nécessaire de décrire ce grand nombre de couche, où chaque paramètre doit être ajusté et testé. Dans le temps imparti pour ce travail de Bachelor, il est difficilement envisageable d’explorer tous ces paramètres et ces complexités. Aussi, des modèles développés de complexités moindre ont été testés, cependant, il a été remarqué que les résultats n’étaient pas concluants.

Il est aussi possible de remarquer qu’aux vues des complexités nécessaires au bon fonctionnement de tels algorithmes, le temps d’entraînement en est de très longue durée. Ainsi, à titre d’exemple, on parle de plus de 3 jours d’entraînements du modèle *RCNN* sur le dataset *Pascal VOC 2007*³⁶ (probablement effectué sur GPU puissant). Un tel entraînement semble difficilement réalisable dans le cadre de ce travail.

C’est pourquoi l’utilisation d’API de détection d’objet a été préférée. En général, des modèles dont les architectures correspondent à *Faster-RCNN* ou *Yolo* et qui ont été pré-entraînés sur des *dataset* sont disponibles. Ceux-ci ont déjà la capacité de détecter des objets. Il est ensuite possible de les utiliser et de les spécialiser afin de détecter n’importe quel objet de notre choix. Procéder ainsi permet de réduire largement le temps d’entraînement nécessaire. Il faut cependant aussi remarquer qu’en conséquence, le principal défaut de cette approche est que l’architecture du modèle ne peut pas être choisie.

4.5.5 API de détection d’objets

L’utilisation d’une API de détection d’objets présente de meilleurs résultats. Ici sont décrit leurs conceptions.

Tensorflow

L’API de détection d’objets de *TensorFlow* a été utilisée.

Concernant la conception d’un modèle, deux choix étaient possibles :

- L’API permet de définir un modèle à partir de 0. Cette méthode peut être utilisée.
- Il est aussi possible d’utiliser des modèles préconçus.

Comme vu précédemment en section 4.5.4 *Limitations d’un développement complet de réseau de neurones*, l’utilisation d’un modèle prédéfini et pré-entraîné est préférable. La figure 4.22 en présente quelques uns disponibles. Ceux-ci ont été entraînés sur le dataset *Coco*³⁷. Le temps de traitement d’une image est indiqué dans la colonne *speed*, quant à *COCO mAP*, il s’agit d’une mesure prenant en compte plusieurs sous-métriques telle que la précision moyenne des *bounding box* des objets³⁸. Plus la valeur *mAP* est grande, meilleure est la détection.

Deux principales architectures peuvent être distinguées :

SSD les temps d’exécutions sont relativement rapides, mais les résultats assez faibles

RCNN les temps d’exécutions sont plus long, mais les résultats meilleures.

36. Fei-Fei Li, Andrej KARPATHY et Justin JOHNSON. *Parking automobiles véhicules*. 1^{er} fév. 2016. URL : <https://web.cs.hacettepe.edu/~aykut/classes/spring2016/bil722/slides/w05-FasterR-CNN.pdf> (visité le 19/07/2018).

37. *Coco* est un grand corpus d’images annotées de divers objets (animaux, véhicules, objets du quotidien, etc.) (MSCOCO. *Coco: Common Object in Context*. URL : cocodataset.org/ [visité le 22/07/2018])

38. Plus d’informations concernant la métrique *COCO mAP* sont disponibles sur <http://cocodataset.org/#detection-eval>

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

FIGURE 4.22 – Capture d’images de plusieurs caméras

Dans notre cas, il n'est pas nécessaire de pouvoir analyser une image rapidement, mais la qualité de la détection doit être bonne.

C'est pourquoi le modèle *faster_rcnn_inception_v2_coco* a été choisi, afin d'obtenir un temps de traitement raisonnable³⁹ et une précision assez bonne. Après quelques recherches, il semble aussi que ce modèle soit assez léger à entraîner, notamment relativement à des modèles *RCNN* "simple" (non désigné par *faster*).

Il aurait été idéal d'évaluer ces différents modèles de manière plus approfondie. Cependant, la mise en œuvre d'un protocole d'évaluation peut paraître compliqué : en effet, certains paramètres qu'il faudrait prendre en compte ne sont pas précisés dans ce tableau, tel que le temps d'entraînement nécessaire avec chacune des architectures. Aussi, pour une évaluation efficace, il serait aussi souhaité d'entraîner

39. Il faut noter que les temps de traitement indiqués sont sans doute effectués sur GPU. Sur processeur, les temps peuvent être largement multipliés.

ces différentes architectures sur un corpus d'images de parking. A la vue des temps d'entraînements long, cette approche est difficilement réalisable.

Yolo

L'architecture *Yolo* a aussi été testée. Il en existe plusieurs versions. La dernière en Juillet 2018, *Yolo v3*, a été préférée. Elle apporte des meilleures performances par rapport aux versions antérieures, tant au niveau des résultats qu'en puissance de calculs nécessaires.⁴⁰

Comme vu précédemment, il est préférable d'utiliser un modèle pré-entraîné. L'implémentation de *Yolo* utilisée en fournit un, entraîné sur le corpus d'image *Coco*.⁴¹ C'est celui qui a été utilisé.

4.6 Evaluation des modèles

Il a été choisi d'évaluer les modèles par rapport aux résultats qui sont finalement attendus, soit le nombre de voitures présentes sur le parking.

En *machine learning*, les méthodes classiques d'évaluations consistent souvent aux calculs des précisions et rappels reposant sur une matrice de confusion. Cependant, c'est ici un problème de régression, où on cherche à approcher au mieux le nombre de véhicules présent sur le parking. Une métrique souvent utilisée est la *RMSE*, pour *Root Mean Squared Error*.⁴²

Pour chaque image du corpus d'évaluation, le nombre de voitures présentes est prédit par l'algorithme (\hat{n}), qui est comparé au nombre de voitures réelles (n), tiré de l'annotation. La formule décrite ci-après présente le calcul effectué :

$$\sqrt{\sum(n - \hat{n})^2}$$

40. REDMON et FARHADI, cf. note 8.

41. PJREDDIE, cf. note 29.

42. WIKIPEDIA. *Root Mean Squared Deviation*. URL : https://en.wikipedia.org/wiki/Root-mean-square_deviation (visité le 23/07/2018).

5 Réalisation

5.1 Disponibilité des fichiers sources

L'entièreté de ce projet est open-source et est disponible sur un *repo Github*, à l'adresse suivante :

https://github.com/remij1/TB_2018

5.2 Structure du projet

En premier lieu, il a été souhaité de définir la structure de dossier de développement du projet (dossier *dev*). Elle est définie ci-après.

tensorflow_models API de détection d'objets de *Tensorflow*

park_python Le développement effectué

camera Connexion à la caméra et récupération automatique d'images

dataset_helper Méthodes d'aides au traitement de *dataset*, comme pouvoir séparer celui en 3 sous dossiers *test*, *dev* et *train*

final_models Modèles finaux utilisés

logger Système de log développé

ml_helper Fonctions d'aide à l'entraînement et à l'utilisation des modèles. Contient notamment des *predictor* permettant de calculer le nombre de voitures présentes en fonction d'une image

drafts Création et entraînement des modèles, traitements d'images

cam_image_processing Test de traitements d'images

edge_detection Détection de bords

image_regression Création et entraînement du modèle approchant le problème sous forme de régression

object_detection Modèles de détection d'objets

darknet Modèle de détection de voiture basé sur *Yolo*

grid_keras Modèle de détection des voitures à l'aide d'une grille et *Keras*

tensorflow_api_cars Modèle de détection d'objets utilisant l'API *tensorflow* sur le corpus *Cars*

tensorflow_api_pklot Modèle de détection d'objets utilisant l'API *tensorflow* sur le corpus *PKLot*

rest_app.py Application permettant d'exposer une API *REST* fournissant à un utilisateur l'état actuel du parking.

5.3 Capture d'images

5.3.1 Emplacement de la caméra

En date du 29 mai 2018, la caméra a été installée, connectée au réseau et est fonctionnelle. La photo présentée en figure 5.1 est une vue aérienne du site de Cheseaux de la HEIG-VD. C'est ici qu'un parking sera filmé.



FIGURE 5.1 – Emplacement de la caméra (en rouge) et du parking filmé (en vert)
Google MAPS. HEIG-VD. 2018. URL : <https://www.google.com/maps/@46.7794439,6.6587222,18z> (visité le 15/04/2018)

Y est désigné par un cercle rouge l'emplacement de la caméra. Elle se situe sur la terrasse du bâtiment est, accessible au niveau K. Elle est orientée afin de pouvoir capturer des images du parking désigné en vert.

Malheureusement, lors des tests effectués après réception de la caméra **Wanscam HW0029-5**, il a été remarqué que le signal Wifi présent sur le toit n'était pas suffisamment fiable afin de connecter la caméra. Afin de palier à ce problème, un câble réseau Ethernet a tout de même pu être tiré de manière temporaire.

5.3.2 Configuration des périphériques

Dans cette sous-section, on trouvera des informations utiles concernant la caméra et la machine virtuelle. Ces informations seront utilisées dans la suite de ce rapport.

Afin de pouvoir accéder à la caméra, un FQDN¹ a été configuré. Celui-ci permet d'obtenir l'adresse IP de la caméra à l'aide du DNS local. Ainsi, plutôt que de s'adresser à elle par une adresse IP, il suffit

1. *FQDN* : Fully qualified domain name, soit nom de domaine entièrement qualifié. (WIKIPEDIA. *Fully qualified domain name*. 25 déc. 2016. URL : https://fr.wikipedia.org/wiki/Fully_qualified_domain_name [visité le 13/06/2018])

d'utiliser en lieu et place l'adresse suivante :

ipcam.einet.ad.eivd.ch

Comme indiqué en section 4.1.1, une VM est mise à disposition de ce projet afin de récupérer des images de la caméra. Son adresse IP, 10.192.75.100, est fixe.

5.3.3 Requêtes et protocole

La caméra **Wanscam HW0029-5** expose un serveur web permettant sa configuration. Bien entendu, celui-ci permet des connexions authentifiées. Il utilise le protocole *Basic Auth HTTP*².³ Il est cependant important de remarquer que le serveur ne permet pas l'utilisation d'*HTTPS* : ainsi, la connexion n'est pas chiffrée. Le mot de passe fournit à la caméra, bien qu'encodé, circule en clair sur le réseau. Il est donc important de noter que dans le cadre d'une application professionnelle, ceci n'est pas envisageable.

Elle expose un *endpoint HTTP* permettant de récupérer l'image actuelle que filme la caméra. Ainsi, dans le cadre de ce projet, l'adresse *http ://ipcam.einet.ad.eivd.ch/web/tmpfs/snap.jpg* est utilisée. La figure 5.2 présente donc ce protocole de communication qui est utilisé afin de récupérer des images.

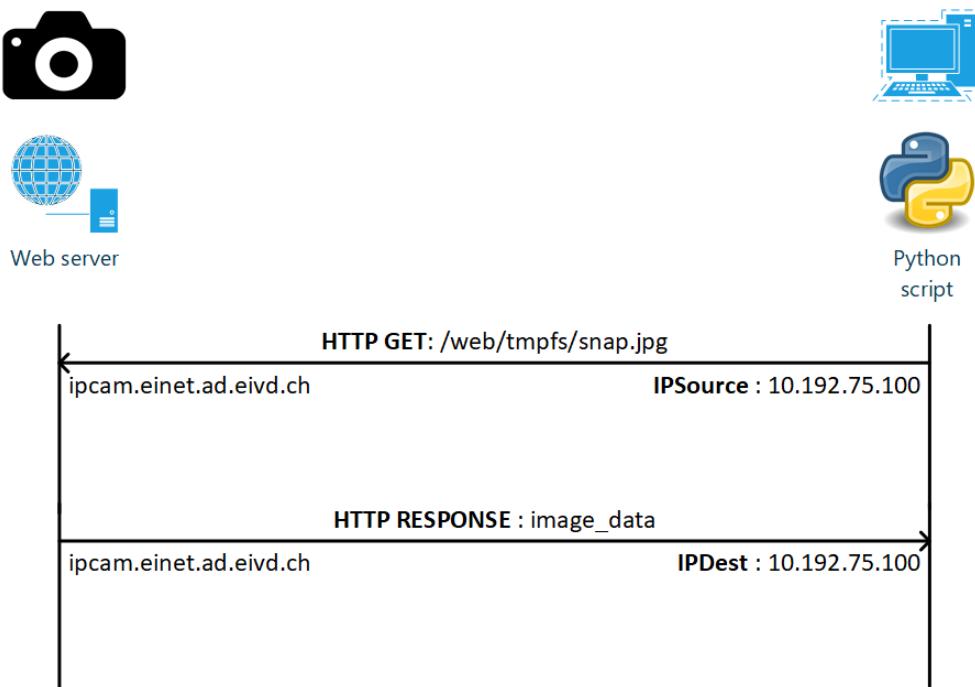


FIGURE 5.2 – Requête d'une image à la caméra **Wanscam HW0029-5**

2. Le protocole Basic Auth consiste à inclure dans les entêtes HTTP un champ *Authorization*. Celui-ci contient le login et le mot de passe de l'utilisateur, sous forme encodée (*Base64*)

3. WIKIPEDIA. *Basic access authentication*. 7 juin 2018. URL : https://en.wikipedia.org/wiki/Basic_access_authentication (visité le 13/06/2018).

5.3.4 Agent

Un agent a été développé, déployé sur la VM. Celui-ci permet de définir un intervalle auquel des images seront demandées à la caméra. Il permet donc de gérer la connexion à celle-ci, mais aussi les pertes de liaisons. A la réception d'une image, il permet de définir une méthode de traitement (décrise en section 4.4 et 5.4). Il permet aussi de définir, si tel est souhaité, une heure de début et une heure de fin durant lesquelles les requêtes seront effectuées. Lorsqu'on sort de cet intervalle, aucune photo ne sera capturée. Dans notre cas, celui-ci est utilisé afin d'éviter une multitude d'images prises du parking vide aux heures de nuit.

On trouvera au listing 5.1 la création en python de celui-ci. Il faut remarquer qu'il n'a pas été souhaité de préciser les constantes USERNAME et PASSWORD pour des raisons de confidentialité. Il est aussi nécessaire de préciser que handle_image est la fonction qui décrit le traitement effectué sur chaque image reçue. Celle-ci sera décrite en section 5.4

Listing 5.1 – Création d'un agent récupérant les images

```

1 CAMERA_HOST = "ipcam.einet.ad.eivd.ch"
2 IMAGE_REQUEST_MIN_DELTA = 60
3 IMAGE_REQUEST_START_TIME = time(4, 30) # Start time at 4h30 AM
4 IMAGE_REQUEST_STOP_TIME = time(23) # Stop time at 11 PM
5 # [...]
6
7 # Creating a camera client
8 camera = CameraClient(CAMERA_HOST, USERNAME, PASSWORD)
9 # Creating a agent which requests the camera for an image once every
   hour
10 agent = CameraAgent(camera, handle_image, minutes =
    IMAGE_REQUEST_MIN_DELTA, running_time=(IMAGE_REQUEST_START_TIME,
    IMAGE_REQUEST_STOP_TIME))

```

Ainsi, une image sera demandée toutes les heures, de 4h30 à 11h.

5.3.5 Monitoring

La caméra utilisée fonctionne sur batterie et panneaux solaires. Ainsi, il a semblé important de pouvoir surveiller le bon fonctionnement du système, et d'être averti en cas de malfonctionnement afin de pouvoir agir au plus vite. On pensera notamment à une perte de connexion due à des batteries faibles (par exemple, suite à un manque de soleil sur plusieurs jours consécutifs), ou encore à un câble déconnecté.

Python offre un système complet natif de journalisation. Lorsque ce système est utilisé dans les modules créés, il est aisément de traiter des logs, et même d'avertir par mail des erreurs produites. Ce module peut être importé grâce à l'instruction `import logging`.

Ce système offre plusieurs niveau de log. On les trouvera ci-dessous, du plus critique au moins important :⁴

— CRITICAL

4. PYTHON.ORG. *Logging facility for Python*. URL : <https://docs.python.org/3/library/logging.html> (visité le 14/06/2018).

- ERROR
- WARNING
- INFO
- DEBUG
- NOTSET

Ainsi, le niveau de chacun des logs effectués lors de la capture d'image a été défini. Une liste des logs importants concernant ce monitoring est proposée ci-dessous :

- INFO** Une image a été capturée et bien reçue
- ERROR** La connexion à la caméra a été perdue
- INFO** La caméra n'est toujours pas accessible
- WARNING** La connexion à la caméra a été rétablie

Python permet d'écouter les logs émis par un module. Il est aussi possible de définir le niveau d'écoute. Par exemple, il peut être souhaité de capturer tous les logs de niveau *WARNING*. Dans ce cas, tous les logs de ce type seront capturés, ainsi que tous les logs dont le niveau est supérieur (soit plus important).

Lors de l'arrivée d'un log, un système de *handler* (proposé par *Python*) a été mis en place. On en distinguera 3 :

- Terminal handler** Ecoute les logs à partir du niveau *INFO*. A l'arrivée d'un log, celui-ci est affiché dans la console. Les informations de debug ne sont pas affichées.
- File handler** Ecoute tous les logs. Ainsi, une trace est gardée de tous les logs dans le système de fichier. Ce gestionnaire est de type *TimedRotatingFileHandler*, qui permet de créer un nouveau fichier de logs par jour.
- SMTPHandler** Ecoute les logs à partir du niveau *WARNING*. Lors d'une erreur ou d'un avertissement, un mail est envoyé automatiquement.

Ainsi, il est possible d'agir rapidement lors d'une perte de connexion à la caméra. Le log associé étant de type (*ERROR*), un mail est envoyé à l'aide du *SMTPHandler*. Si la caméra est reconnectée (*WARNING*), un mail est aussi reçu.

5.4 Traitement des images

La figure 5.3 présente la récupération et le traitement des images qui est effectué.

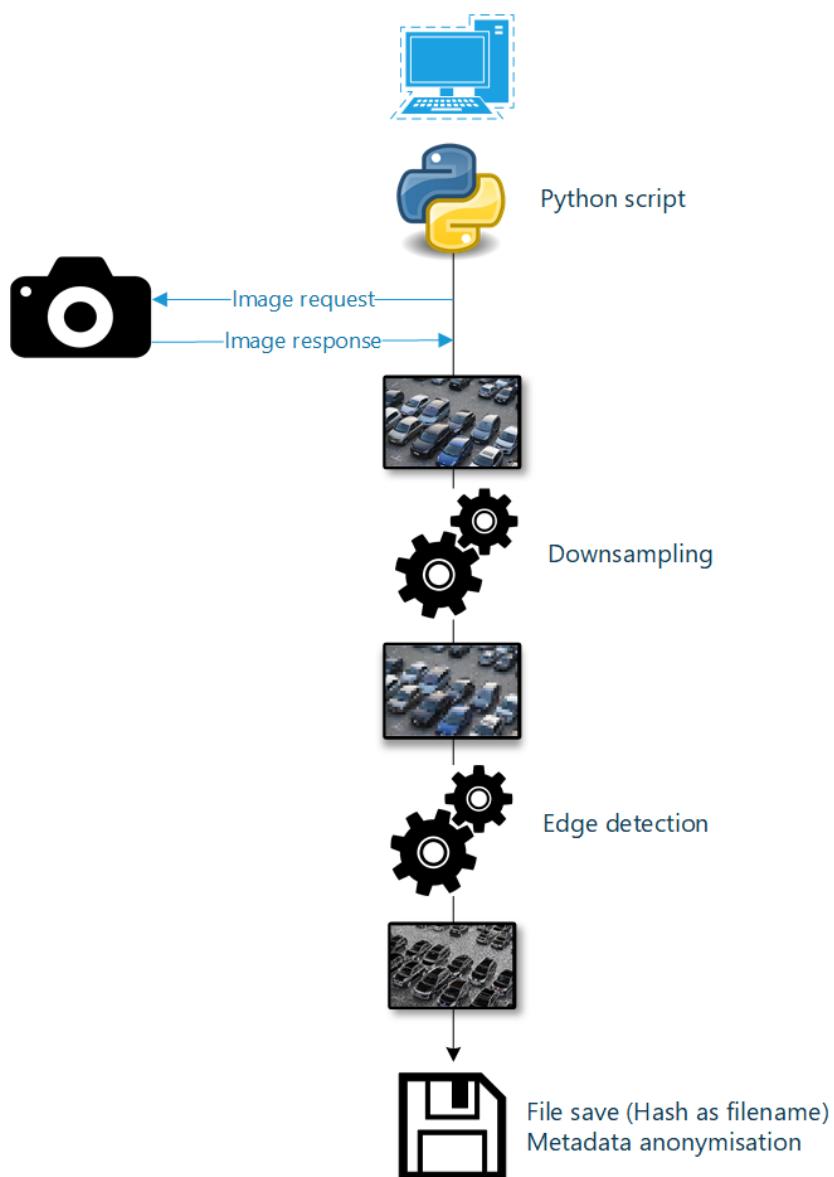


FIGURE 5.3 – Requêtes d’images et traitement

Dans un premier temps, un script Python interroge la caméra et récupère une image tel qu'il a été décrit en section 5.3 précédente. Ensuite, l'image est sous-échantillonnée, et les bords détectés. Pour ce faire, *Scikit-image* a été utilisé.

Le code 5.2 présenté montre la méthode traitant les images entrantes. Celles-ci sont passées en argument de la méthode (`image_stream`).

Listing 5.2 – Traitement des images reçues

```

1 # Defining what to do when an image is received
2 def handle_image(image_stream):
3     # Converting to a skimage/opencv image (simply a [x, y, 3] numpy
4     array)
5     image = io.imread(image_stream)

```

```
6      # Firstly, downsampling the image
7      image = transform.resize(image, IMAGE_OUTPUT_SIZE, mode='reflect',
8                                anti_aliasing=True)
9      # Secondly, detecting the edges
10     image = scharr_hsv(image)
11
12     # Then, we output the image to the folder
13     # We use a hash as an ID
14     # We could have used a datetime format as a unique filename, but
15     # this could results to a lack of privacy
16     # Saving it as a bitmap: no decompression to do when loading a
17     # lot of file
18     # Note: hashing time is negligible
19     h = hash(image.data.tobytes())
20     filename = IMAGE_FOLDER + hex(h)[-15:] + ".bmp"
21
22     with warnings.catch_warnings(): # used to ignore loss of
23       precision warning
24       warnings.simplefilter("ignore")
25       io.imsave(filename, image)
26
27     # We then have to reset metadata access and edit time because of
28     # privacy issues
29     os.utime(filename, (946684800, 946684800))
```

La méthode `_scharr_hsv` utilisée est décrite au listing de code 5.3.

Il est possible d'y distinguer la ligne `@adapt_rgb(hsv_value)`,

provenant de la librairie *scikit-image*. Elle permet d'adapter le filtre *scharr* présent dans la méthode sur chacun des canaux de l'image.

Listing 5.3 – Détection de bord à l'aide de *scikit-image*

```
1 @adapt_rgb(hsv_value)
2 def _scharr_hsv(image):
3     return filters.scharr(image)
```

Dans le cadre de l'anonymisation des données, il a semblé important qu'aucune information concernant les utilisateurs puisse apparaître. C'est pourquoi le nom de l'image est basé sur son hash (fonction de hash fournie par Python. Un hash complexe n'est pas nécessaire).

L'image est ensuite sauvegardée à un format *bitmap* (*.bmp*), sans pertes de compression. La date et l'heure de création et de modification du fichier sont aussi modifiées pour que les données soient au plus anonymisées.

5.5 Traitement du corpus *PKLot*

Les images fournies par *PKLot* sont annotées par place de parc marquée au sol. Cependant, il est possible que certaines voitures ne soient pas stationnées sur l'une de celle-ci.

Si lors de l'entraînement, l'image segmentée par place de parc est utilisée, cela ne poserait aucun problème. Cependant, dans le cadre de ce projet, les modèles sont entraînés sur l'ensemble de l'image. Ainsi, certaines voitures présentes n'ont pas été annotées comme telles. Si ce *dataset* n'était pas traité, certaines voitures seraient considérées comme étant de l'arrière-plan, et ne seraient pas détectées.

Le corpus *PKLot* est séparé en 3 dossiers différents pour chacune des caméras disponibles. Chaque image a été traitée à l'aide d'une méthode de remplissage, permettant d'effacer certaines régions d'une image.⁵ Pour ce faire, *OpenCV* a été utilisé. La figure 5.4 en présente un résultat.

Il faut noter qu'un remplissage de qualité n'est pas nécessaire. En effet, il est avant tout nécessaire d'effacer les voitures présentes qui n'ont pas été annotées comme telles.

Par la suite, les entraînements qui ont été effectués utilisent l'ensemble de ces images traitées, sans distinction entre les caméras. Cela permet d'augmenter la généralisation des modèles générés.

5.6 Mise en place et entraînements des modèles

5.6.1 Machine virtuelle

Afin d'obtenir de meilleures performances pour l'entraînement des modèles, une machine virtuelle a été créée sur *AWS*⁶. Elle fournit aussi des images d'instances spécialisées dans le *Machine learning*, où par exemple *TensorFlow* et *Keras* sont préinstallés⁷. C'est celle-ci qui a été utilisée. Les corpus d'images nécessaires à l'entraînement y ont été téléchargés et pré-traités à l'aide de scripts *Python* qui ont été développés dans le cadre de ce projet.

5.6.2 Format *VOC XML*

Un format courant d'annotations d'images est le format *VOC*, qui permet une certaine interopérabilité. Un exemple en est donné au code 5.4. Le nom de l'image ainsi que sa taille y est indiqué, Aussi, chaque objet présent dans l'image y est indiqué, ainsi que sa classe et sa *bounding box* associée. Il y a un fichier *VOC* par images.

Listing 5.4 – Exemple de fichier *VOC*

```

1 <annotation>
2 <filename>2012-11-10_07_12_40.jpg</filename>
3 <size>
4   <width>1280</width>
5   <height>720</height>
6   <depth>3</depth>
7 </size>
```

5. WIKIPEDIA. *Inpainting*. URL : <https://en.wikipedia.org/wiki/Inpainting> (visité le 24/07/2018).

6. AWS, pour *Amazon Web Service*, fourni un système de IaaS (*Infrastructure as a Service* où il est possible, entre autres, d'instancier des machines virtuelles puissantes). <https://aws.amazon.com/>

7. Les images sont disponibles à l'adresse <https://aws.amazon.com/fr/machine-learning/amis/>.

```

8 <segmented>0</segmented>
9 <object>
10   <name>car</name>
11   <bndbox>
12     <xmin>669</xmin>
13     <ymin>419</ymin>
14     <xmax>717</xmax>
15     <ymax>475</ymax>
16   </bndbox>
17 </object>
18 </annotation>
```

Les annotations fournies par *PKLot* sont dans un format *XML* qui leurs sont propres. Il a été nécessaire de transcrire ces *XML* au format *VOC*. Pour ce faire, un script *Python* a été utilisé. Celui-ci utilise la librairie *etree* pour *parse* le fichier original, et en créer un nouveau correspondant au format *VOC* voulu.

5.6.3 Tensorflow Object Detection API

Comme précédemment indiqué dans ce rapport, l'API de détection d'objets fournie par *Tensorflow* a été utilisée. Celui-ci n'étant pas inclus dans *TensorFlow*, le *repo*⁸ de l'API a été cloné et utilisé. Il a été nécessaire de l'ajouter aux variables d'environnements *Python* afin qu'il soit possible d'importer directement les librairies nécessaires depuis un script *Python*.

Afin d'utiliser un corpus d'entraînement à l'aide de *Tensorflow*, des fichiers *.record* doivent être générés. Ceux-ci contiennent les images et leurs annotations. Ils peuvent être aisément créés à partir de fichiers *VOC* à l'aide de la librairie *TensorFlow* et d'un script *Python*.

Un fichier de configuration (*.config*) est aussi nécessaire. Celui-ci indique où se situe les fichiers *.record*, le modèle pré-entraîné, etc. Par la suite, l'entraînement est exécuté en ligne de commande.

Listing 5.5 – Exécution d'un entraînement à l'aide de *Tensorflow Object Detection API*

```
python /home/ubuntu/tensorflow_models/research/object_detection/train
      .py \
--logtosterr \
--train_dir=/home/ubuntu/DS/PKLot/tensorflow_ds/training/ \
--pipeline_config_path=/home/ubuntu/TB/GIT/dev/park_python/drafts/
          object_detection/tensorflow_api_pklot/training.config
```

5.6.4 Yolo

Une implémentation de *Yolo* a aussi été utilisée. Pour ce faire, il a été nécessaire de télécharger le code source du réseau de neurones open-source *Darknet*,⁹ en clonant le *repo* associé.

Une fois celui-ci téléchargé, l'attribut *OPENMP* du fichier *Makefile* a été défini à 1 afin que les entraînements puissent être effectués sur tous les coeurs de la machine disponible. Puis, le code a été compilé à l'aide de la commande *make linux*. Il faut noter que cette commande n'est pas nativement disponible sous Windows.

8. On trouvera le repo sur https://github.com/tensorflow/models/tree/master/research/object_detection

9. PJREDDIE, cf. note 29.

Dans un second temps, il est nécessaire de préparer les labels pour l'entraînement. Puisque les formats *VOC* ont été créés pour le corpus *PKLot*, un script *Python* fourni par *Darknet* a été utilisé. Les fichiers de configurations nécessaires à l'entraînement sur le *dataset* ont aussi été modifiés.

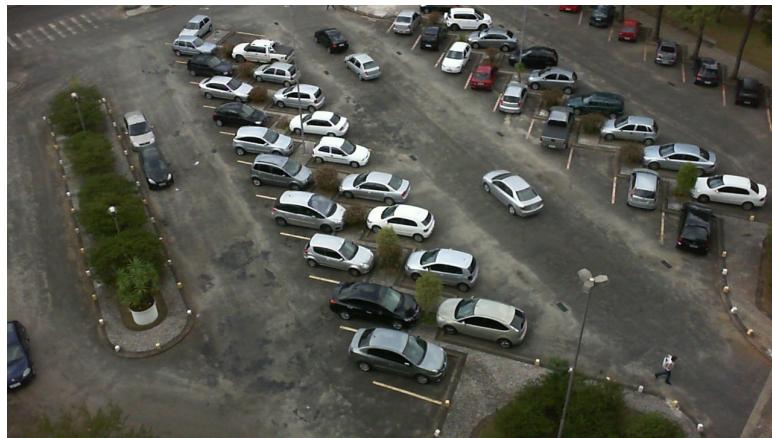
Listing 5.6 – Exécution d'un entraînement à l'aide de *Yolo*

```
./ darknet detector train cfg/coco.data cfg/yolov3.cfg darknet53.conv  
.74
```

5.7 Utilisation du modèle final *Tensorflow*

Comme l'indique les tests en section 6.3 *Choix final*, un modèle utilisant l'API de *Tensorflow* a été choisi.

Une classe, *TensorflowPredictor*, a été créée, basée sur des codes d'exemples disponibles (dont les sources sont précisées dans le code source). Lors de son instanciation, le chemin vers le modèle qu'on souhaite utiliser est passé en paramètre, ce qui a pour effet de le charger en mémoire. Ensuite, il est possible d'utiliser cet objet afin de prédire le nombre de voitures présentes sur une image. Le script permet aussi une exécution en ligne de commande afin de pouvoir générer les *bounding box* d'une image.



(a) Avant remplissage



(b) Masque utilisé



(c) Après remplissage

FIGURE 5.4 – Avant-après traitements d'une image du corpus *PKLot*

6 Tests et validation

Ce chapitre permet d'évaluer et comparer les modèles de détections d'objets *faster RCNN* et *Yolo v3*, tels que décrits en section 4.5.5 *API de détection d'objets*. Il est important de noter qu'il a été remarqué qu'en général, la détection d'objets repose sur un entraînement à l'aide des images où la détection de bord n'a pas été effectuée. Aussi, quelques entraînements ont été effectués sur les images traitées, qui n'ont apportés aucun résultat concluant. C'est pourquoi seul le corpus *PKLot* (pré-traité à l'aide d'un remplissage de région tel qu'indiqué en section 5.5) est utilisé.

Ont été évalués :

- Le modèle pré-entraîné.
- Le modèle ré-entraîné sur le corpus d'image *PKLot* avec différents nombres d'itérations. Le nombre d'itérations nécessaires diffère en fonction des modèles utilisés.

Chacun des modèles pré-cités a été évalué à l'aide de la méthode décrite en section 4.6 *Evaluation des modèles*.

6.1 TensorFlow Object Detection API

Le modèle *Faster RCNN* a été évalué à l'aide de l'API de détection d'objets de *TensorFlow*. On trouvera en figure 6.1 l'entraînement qui a été effectué sur le corpus d'images *PKLot*.

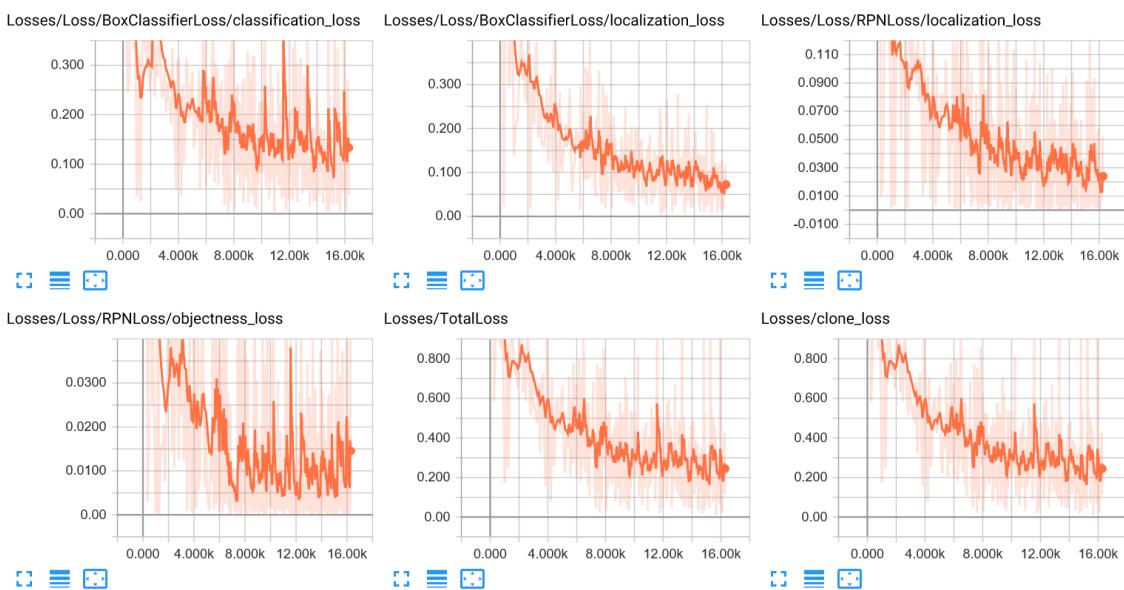


FIGURE 6.1 – Entrainement sur le corpus *PKLot* (graphes par *Tensorboard*)

On distingue plusieurs fonctions objectifs :

Classification loss La perte au fil des itérations lors de la classification.

Localization loss Présente à quel point les *bounding box* désignant les voitures sont bien localisées.

Objectness loss Pour chaque *bounding box*, il s'agit de définir si c'est une voiture ou l'arrière-plan

Total loss Présente une combinaison des toutes les fonctions de pertes.

L'entraînement a été stoppé après 16000 itérations (plus de 2 jours de calculs). En effet, il semble qu'un certain palier ait été atteint. Aussi, plus d'itérations pourraient entraîner à un sur-apprentissage. Les nombres d'itérations 4000 et 16000 ont été évalués, ainsi que le modèle pré-entraîné.

Afin d'effectuer les tests, il est nécessaire de choisir un score minimum. Celui-ci indique, pour chaque objet détecté, à quel point l'algorithme est confiant sur sa prédiction. La figure 6.2 présente une détection effectuée sur une image du *dataset PKLot*.

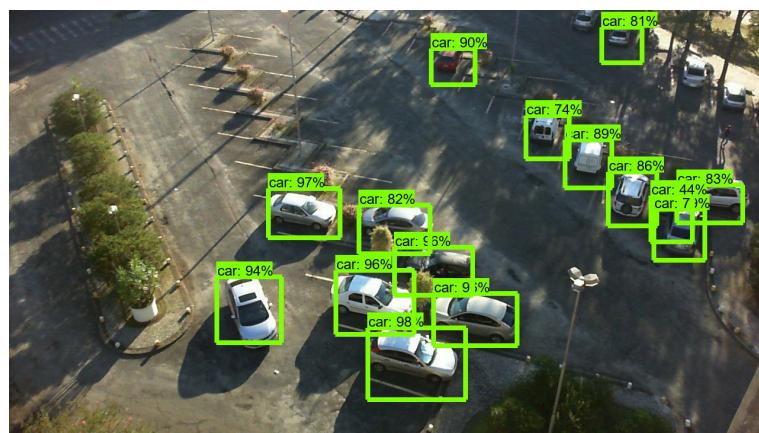


FIGURE 6.2 – Exemple de détection sur un entraînement de 4000 itérations

Les scores sont généralement supérieurs à 75%. Il est possible de remarquer un faux positif d'un score de 44%. Un seuil de 70% a été défini, afin d'obtenir une certaine marge, et réduire le nombre de faux positifs. Cependant, il faut remarquer que si le modèle *Faster RCNN* est utilisé, ce seuil peut être abaissé à 20% pour obtenir des résultats raisonnables. Ceci permet aussi d'indiquer que l'utilisation d'un modèle non ré-entraîné ne permet pas une grande confiance en l'algorithme.

Résultats

Le tableau 6.1 présente les résultats du modèles *Faster RCNN* pré-entraîné (0 itérations), ainsi qu'en-traîné avec 4000 et 16000 itérations supplémentaires sur le corpus d'images *PKLot*. Les calculs qui ont été effectués ont été décrits en section 4.6 *Evaluation des modèles*.

Cars dataset

Un entraînement sur le *dataset Cars*, vu en section 4.3 *Corpus d'images*, a aussi été testé. Cependant, les résultats obtenus n'étaient pas bons, où un entraînement entraînait des résultats moins bons que le modèle pré-entraîné. Il est possible que ce corpus ne soit pas adapté pour des images de parkings, car les angles de vues prise des voitures et leurs tailles n'est pas similaire. Il n'a donc pas été souhaité de préciser les détails des résultats ici.

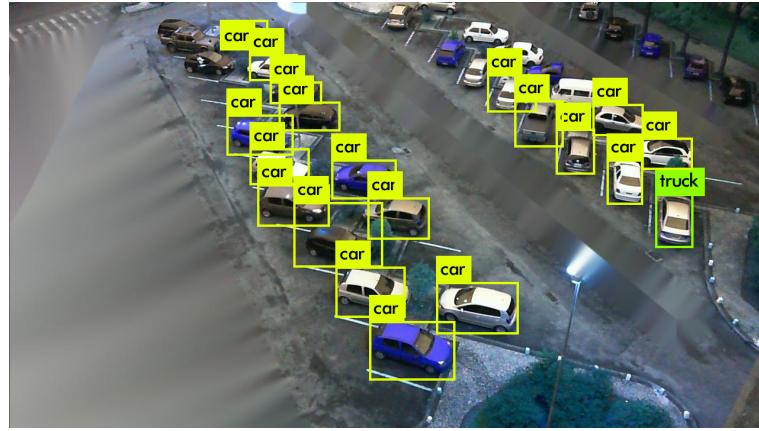
Le modèle entraîné 4000 fois semble le meilleur lors de l'évaluation. La cause est sans doute un sur-entraînement du modèle à 16000 itérations, ce qui ne permet pas de généraliser suffisamment.

N° itération	RMSE
0	32.06564071
4000	10.4427355
16000	22.88589282

TABLE 6.1 – *Faster RCNN* - Résultats

6.2 Yolo v3

Le modèle *Yolo* a aussi été évalué. Un modèle pré-entraîné sur le corpus d’images *Coco*¹ à été utilisé.

FIGURE 6.3 – Exemple de détection avec un modèle *Yolo v3* pré-entraîné sur *Coco*

Le réseau de neurones a aussi été entraîné sur le *dataset PKLot*, jusqu’à un nombre de 200 itérations. Cependant, les résultats n’ont pas été concluants, où seules quelques voitures ont pu être détectées. 200 itérations semble un nombre très faible. Cependant, le temps d’entraînement nécessaire était trop grand, où moins de 100 itérations par jours étaient effectuées. Ainsi ici, aux vues des résultats médiocres, il n’a pas été souhaité de les intégrer à cette évaluation. Ainsi, seul le modèle pré-entraîné sur le corpus d’images *Coco* a été effectué.

De la même manière qu’auparavant, l’erreur au carré moyenne a été calculée sur le nombre de voitures détectées. On en trouvera le résultat en figure 6.2.

Modèle	RMSE
Yolo v3 entraîné sur <i>Coco</i>	34.75053725

TABLE 6.2 – *Yolo* - Résultats

6.3 Choix final

A la vue des résultats précédents, il semble que le meilleur algorithme soit le modèle *Faster RCNN* ré-entraîné sur *PKLot* avec 4000 itérations supplémentaires. Il faut noter que ces résultats ont été

1. MSCoco, cf. note 37.

obtenus à l'aide du corpus d'images *PKLot* : ainsi, certains paramètres, comme le seuil de détection, doit peut-être être modifié.

6.4 Tests en conditions réels

Une fois le système mis en place (caméra, prédiction du taux d'utilisation, *REST API*), il aurait été souhaité d'effectuer une évaluation approfondie de la détection des voitures. Cependant, pour cause de protection des données des utilisateurs du parking, seul quelques observations ont pu être effectuées.

En général, il a pu être observé que le système semble viable, où seules 1 ou 2 voitures ne sont pas détectées. Il a pu être remarqué que le modèle ne détecte que très mal les voitures présentes sur le bord de l'image, où seul une partie de celle-ci est visible.

Aussi, comme on pouvait s'y attendre, les voitures présentes derrière les arbres ne sont que très rarement détectées, ceci dû à la verdure qui les cache. Ce n'est pas un problème en soi : il suffit de considérer le parking derrière les arbres comme ne faisant pas parti du parking dont on souhaite connaître l'état. S'il est tout de même souhaité que ces voitures doivent être détectées, il serait possible d'ajouter une seconde caméra.

7 Conclusions

7.1 Bilan du projet

7.1.1 Résultats obtenus

L'objectif principal, soit de pouvoir mesurer le taux d'occupation d'un des parkings de la HEIG-VD, a été rempli. En général, la plupart des véhicules présents sont bien détectés. Evidemment, tout algorithme de *machine learning*, tout comme l'humain, ne peut prédire correctement dans 100% des cas.

Les résultats obtenus sont donc assez bons. Cependant, quelques voitures ne sont parfois pas détectées, ce qui peut poser des problèmes pour les utilisateurs. Par exemple, lorsqu'une personne souhaite stationner son véhicule dans le parking, si l'application développée indique une place libre alors qu'il n'en reste aucune, le conducteur s'en retrouve frustré. Aussi, le parking pourrait être indiqué comme plein, alors que ce n'est pas le cas. Ces points mis de côtés, le système reste suffisamment précis pour indiquer s'il est possible de stationner, et ce même s'il ne reste que 2-3 places libres. Aussi, si les voitures sont stationnées généralement de manière répartie sur tout le parking, une seule caméra bien placée est suffisante pour faire transparaître le taux d'utilisation global. Sur le site de Cheseaux de la HEIG-VD, la caméra filme un parking près d'une entrée secondaire. Les modifications de son taux d'occupation peuvent sans doute être appliquées au taux d'utilisation du parking principal.

Aussi, une utilisation possible de ce système qui semble tout à fait appropriée est la création de statistiques. Grâce à un tel système, à l'aide d'une simple caméra, des statistiques détaillées concernant les pics d'utilisations, les heures creuses, les heures d'arrivées des premières voitures, etc. peuvent être déduites. Dans un tel contexte, il n'est même pas nécessaire que toutes les voitures soient détectées.

Si le nombre de places libres doit être extrêmement précis, on conseillera cependant une approche basée sur la segmentation des places de parcs, qui semble apporter des meilleurs résultats. Dans ce travail, les voitures stationnées en bords de routes (parcage "sauvage") ont été prises en compte. Cependant, ce système permet de compter les véhicules présents, mais ne peut pas compter les places libres. En effet, s'il y a un manque de marque au sol, les places disponibles varient en fonction de la façon dont sont stationnés les véhicules. Ainsi, pour une mesure précise, le parcage sauvage devrait être interdit.

Finalement, il est souhaité de préciser que les résultats obtenus ont été réalisés sur des images de parkings de jour, en printemps-été. Ainsi, il aurait été intéressant de tester ce système de nuit, avec de la neige, etc. Cependant, il semble qu'avec un nombre suffisant d'images de parkings labélisées sous toutes ces conditions, les résultats qu'il serait possible d'obtenir devraient être plus ou moins équivalents à ceux obtenus dans ce projet.

7.1.2 Réalisation

Le projet s'est dans l'ensemble bien déroulé, où un résultat final a pu être fourni. Quelques points concernant sa réalisation sont précisés ici.

La recherche et le choix de la caméra n'ont pas été faciles. Les modèles de caméras disponibles, les protocoles utilisés, la qualité des caméras, ou encore les systèmes de panneaux solaires sont tant de paramètres qu'il a fallu appréhender. De plus, le fait qu'un système autonome était nécessaire a réduit le nombre de caméras disponibles, dont leurs réelles qualités n'étaient pas assurées. Aucune des marques ne semblait être reconnue, contrairement aux systèmes classiques (non autonome) de caméras de sécurité. Malgré ces difficultés, la caméra *Wanscam* commandée a tenu ses promesses : il n'a été détecté qu'un très faible nombre d'interruptions de quelques heures sur plusieurs mois. Il a cependant été nécessaire, une seule fois, de redémarrer la caméra manuellement.

L'installation de la caméra ne s'est pas déroulée tout à fait comme prévu. En effet, il a fallu résoudre plusieurs problèmes : pas de connexion Wifi entreprise, mot de passe du Wifi ne pouvant contenir certains caractères spéciaux, signal trop faible depuis l'extérieur du bâtiment, câblage de la caméra, temps qui a été nécessaire à son installation, etc. La majeure partie des heures prévues pendant le semestre pour le travail de Bachelor a été d'installer la caméra et de résoudre les problèmes, où aucun *machine learning* n'a pu être effectué.

Il semble important de noter que les images qui ont été traitées et capturées par la caméra tout le long du semestre dans le but de créer un corpus annoté n'ont finalement pas été utilisées pour entraîner l'algorithme de détection d'objets, car le traitement effectué n'améliorait pas les résultats. Cependant, le système de capture de photos automatique développé dans ce but a pu être réutilisé pour le développement de l'API *REST*.

Python 3 a été utilisé dans ce projet. Bien qu'utile pour faire des scripts, la mise en place des librairies externes, comme *Tensorflow Object Detection API* qui a été clonée ou le module *Logging* développé, est parfois fastidieuse. Avec ces librairies, le système d'import n'est pas idéal : il est souvent nécessaire de modifier les variables d'environnements *Python* pour que les importations soient possibles.

Il a été remarqué que pour la détection d'objets ou les réseaux de neurones à convolutions profonds et complexes en général, les temps d'entraînements nécessaires peuvent être extrêmement longs. Tout le processus, du traitement des données aux premiers résultats, prends du temps. Il aurait été préférable de tester beaucoup de modèles différents, ce qui n'était pas possible dans le laps de temps imparti. Aussi, plusieurs modèles ont été définis qui, au final, n'ont apporté aucun résultat. Pour des évaluations complètes, plus de temps aurait été nécessaire.

Une planification a été définie en début de projet. Cependant, elle n'a pas pu être tenue. D'une part, les problèmes survenus lors de l'installation de la caméra ont retardé le début de la phase de *machine learning*. De plus, beaucoup de recherches et de tests ont du être effectués dans le domaine de la détection d'objets avant qu'un modèle fournisse un résultat concluant. On en retirera que, dans un tel domaine où la recherche est en constante évolution et n'est pas entièrement finalisée, la plupart du temps nécessaire à un développement d'un projet concret est consacré à la recherche d'informations.

7.1.3 Améliorations possibles

Le projet en l'état peut être grandement amélioré et de nouvelles fonctionnalités pourraient être développées.

Dans un premier temps, il semble que des évaluations plus approfondies des différents modèles, auraient pu être effectuées. Par exemple, seuls quelques nombres d'itérations différents ont été évalués. Il aurait été idéal de pouvoir suivre les résultats des évaluations au fil des itérations. Aussi, une évaluation plus précise, soit au niveau des voitures détectées plutôt qu'à leur nombre, aurait pu être intéressante. De plus, il aurait été idéal d'évaluer plus amplement les images provenant de la caméra du parking de

la HEIG-VD. Pour ce faire, les images capturées auraient pu être prédites par le modèle choisi. En parallèle, les images sont traitées à l'aide d'une détection de bords et enregistrées. Elles peuvent être ainsi annotées sans nuire à la protection des données des utilisateurs.

Des tests unitaires du système auraient pu être effectués afin de valider le bon fonctionnement du code développé. Cependant, l'accent a été mis sur le sujet principal de ce travail, soit de construire un modèle viable.

Une application utile de ce projet est la création de statistiques d'utilisation du parking. Il serait donc idéal de le montrer par le développement d'un tel système, où le taux d'utilisation du parking serait montré au fil des heures et des jours. Des graphiques pourraient être utilisés.

Une API *REST* simple a été développée. Cependant, elle pourrait être améliorée et agrémentée de quelques statistiques. Aussi, l'utilisateur lambda devrait pouvoir accéder aux informations sur l'état actuel du parking de manière simple. Pour ce faire, on pensera notamment à une interface web, une application mobile, ou encore à un *chatbot* (par exemple *Telegram*) pouvant indiquer à un utilisateur le nombre de places libres. Toutes ces applications peuvent utiliser l'API *REST* développée.

Comme indiqué précédemment en sous-section 7.1.1, les statistiques d'une sous-section de parking peuvent sans doute être appliquées à tout le parking. Cette approche mériterait d'être approfondie.

Plusieurs caméras pourraient être installées pour connaître le taux d'utilisation du parking global. En conséquence, il faudrait gérer les problèmes des voitures présentes sur plusieurs caméras, etc. Une approche possible serait de n'utiliser qu'une sous-région bien définie de l'image capturée par une caméra, afin qu'il n'y ait pas de superposition possible. Il doit aussi pouvoir être possible de détecter si une voiture est la même sur deux images différentes. Pour ce faire, un réseau de neurones à convolutions doit pouvoir être utilisé.

Le problème du parage en bord de route a déjà été abordé. Une solution qui pourrait être explorée afin d'obtenir un taux d'utilisation du parking précis est de détecter algorithmiquement des emplacements libres (sans voiture présente), où la place serait suffisante pour y stationner. Avec un nombre conséquent d'images du parking sous différentes configurations de stationnement, il semble possible de faire correspondre le *pattern* courant à un de ceux présents dans le corpus. Une application indiquant à l'utilisateur où se situent les places libres sur une carte pourrait en conséquence être développée.

7.2 Bilan personnel

Dans l'ensemble, je suis satisfait du travail qui a été effectué et des résultats produits. Finalement, une application concrète, bien que simple, a pu être développée malgré le long cheminement nécessaire et les embûches rencontrées pour arriver jusque là.

J'ai beaucoup appris, tant au niveau technique qu'organisationnel. J'ai pu approfondir largement ma connaissance du langage *Python* et de multiples bibliothèques d'apprentissage automatique (*Keras*, *Tensorflow*, *Darknet*, etc.). Le traitement d'image est un point qui me semble aussi très important pour un développeur, et j'ai, grâce à ce projet, pu l'aborder.

Si ce projet serait à refaire, je l'aborderais cependant de manière différente. Par exemple, plutôt que de rechercher premièrement à résoudre le problème de façon manuelle à l'aide d'un simple réseau de neurones à convolutions, j'essaierai d'utiliser les bibliothèques de détection d'objets disponibles. Dans tous les cas, je trouve intéressant d'avoir pu explorer d'autres façons de faire, comme traiter le problème sous

la forme d'une simple régression, et d'en tirer des conclusions sur les complexités qui sont nécessaires à un tel algorithme.

J'ai suivi le cours de *machine learning* à la HEIG-VD et j'ai observé une très nette différence par rapport à ce projet. En effet, les laboratoires créés pour le cours sont fait pour fonctionner. Ici, dans le cadre du projet, beaucoup d'idées n'ont apportés aucun résultat. On ne sait jamais à l'avance si ce qui est développé fonctionnera. Parfois, le *machine learning* peut donc être frustrant, et le développement de nouvelles idées entre dans le cadre de la recherche scientifique plutôt que dans l'ingénierie.

Pour conclure, je retire de ce travail de Bachelor un bilan très positif. Le projet a été intéressant à réaliser, et les résultats sont impressionnantes. Des voitures peuvent être détectées à partir d'une image, ce qui semblait inimaginable il y a encore quelques années. Le *machine learning*, de jour en jour, apporte des résultats de plus en plus probants. Je suis donc heureux d'avoir pu effectuer mon travail de Bachelor sur un tel sujet, qui me paraît si important pour le futur.

A Résultats des traitements d'images

A.1 Sous-échantillonnages



(a) 1280 x 720



(b) 1120 x 630



(c) 960 x 540



(d) 800 x 450



(e) 640 x 360



(f) 480 x 270



(g) 320 x 180



(h) 160 x 90

FIGURE A.1 – L'image originale redimensionnée à différentes résolutions

A.2 Détection de bords, sous-échantillonnage

Ici, il est considéré que la détection de bord a été effectuée en premier, à l'aide d'un filtre *Scharr* est appliquée sur l'espace HSV de l'image.



FIGURE A.2 – *Scharr* HSV : image utilisée afin d'explorer les différents sous-échantillonnages



FIGURE A.3 – L'image dont les bords ont été détectés, redimensionnée à différentes résolutions

La figure A.3 applique plusieurs sous-échantillonnages de l'image A.2.

Il semble que l'image la plus légère et suffisamment précise afin de pouvoir détecter sans encombre

des voitures est celle dont la résolution est de 480 x 270 pixels. Les images dont la taille est inférieure sont trop floues, où certaines voitures semblent parfois même disparaître.

A.3 Détection de bords, sous-échantillonnage

Ce paragraphe présente les résultats d'un traitement qui consiste en sous-échantillonner l'image dans un premier temps, puis d'en détecter les bords. Lors des différents tests effectués, il a été remarqué qu'une résolution de 320x180 pixels, tel qu'indiqué dans le paragraphe *Sous-échantillonnage*, ne produisait pas des résultats concluants. C'est pourquoi la résolution de 480 x 270 pixels a été préférée. L'image utilisée est présentée en figure A.4.



FIGURE A.4 – 480 x 270 : image utilisée afin d'explorer les différentes méthodes de détection de bords

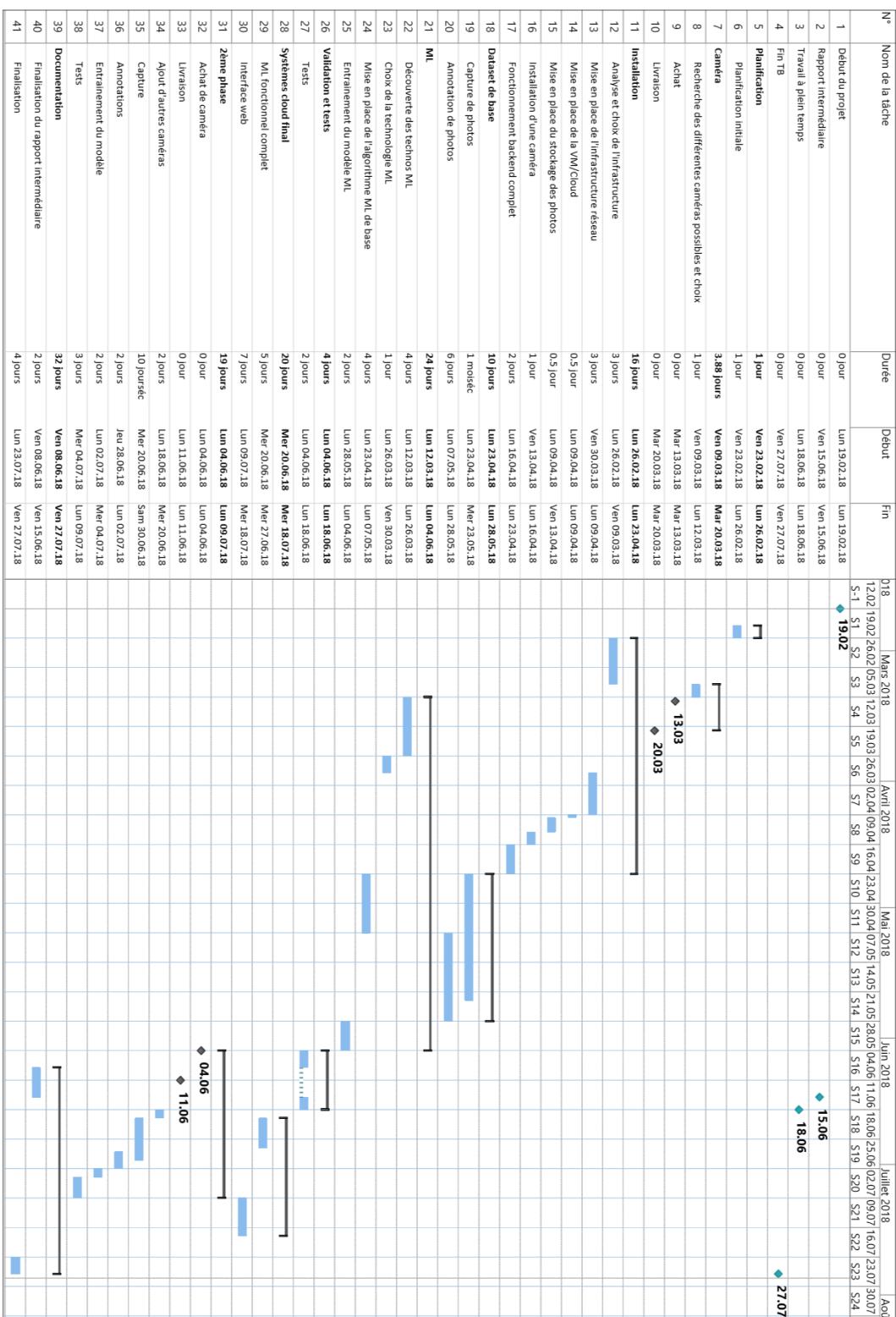
(a) Filtre *Sobel* - RGB(b) Filtre *Scharr* - RGB(c) Filtre *Sobel* - HSV(d) Filtre *Scharr* - HSV(e) Filtre *Sobel* - Valeurs de gris(f) Filtre *Scharr* - Valeurs de gris

FIGURE A.5 – L'image sous-échantillonnée avec différentes détections de bords

La figure A.5 présente l'application des différents filtres sur l'image sous-échantillonnées

Ici, de la même manière que décrit en annexe précédente *Détection de bords, sous-échantillonage*, le filtre semblant le mieux convenir est la matrice de convolution définie par *Scharr* sur l'espace HSV. Les voitures y sont mieux distinguables, notamment grâce à leurs couleurs.

B Planification



C Réalisation

Tâches	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
<i>Mise en place du projet</i>																							
<i>Mise en place caméra</i>																							
<i>Capture et traitement d'images</i>																							
<i>Machine learning - regression</i>																							
<i>Machine learning - grille</i>																							
<i>Machine learning - tensorflow object detection</i>																							
<i>Machine learning - Yolo</i>																							
<i>Rapport</i>																							

Outils utilisés

Dans le cadre de ce travail de Bachelor, plusieurs outils ont été utilisés à l'aide au développement et à la rédaction. On les trouveras dans ce chapitre.

Visual Studio Code

*Visual Studio Code*¹ est un éditeur Open Source comparable à *Sublime Text* ou *Atom*.

Il est possible d'y ajouter des extensions afin de convenir à l'usage souhaité. Dans ce projet, cet éditeur a été utilisé avant tout afin de coder en *Python* et rédiger en *LaTex*. Les extensions suivantes ont donc été installées :

Python Permet d'intégrer à VSCode le développement en Python. Propose un système de *Lintering*, *debugging*, ou encore d'auto-complétion.

autoDocstring Permet de générer automatiquement de la documentation python (*Python Docstrings*)

LaTex Workshop Permet de compiler un projet *LaTex*, proposant de la coloration de code, ou encore un système de *preview* automatique.

BibTexLanguage Permet d'ajouter au fichier de bibliographie *.bib* une coloration syntaxique.

Spell Right Permet la correction orthographique multi-langues de fichiers, dont les documents *LaTex*.

VIM

VIM est un éditeur de texte en ligne de commande. Il a avant tout été utilisé afin d'effectuer des modifications mineurs de code sur la VM.

Conda

Ce gestionnaire d'environnement a été choisi afin de pouvoir interpréter du *Python* facilement. La commande *pip install* a pu être utilisée afin d'installer aisément des librairies supplémentaires. Il faut noter que cette commande est plus du ressort de *Python* que de *Conda*

Git

Git est un gestionnaire de version distribué Open Source. Grâce à celui-ci, il a été possible de suivre l'évolution de la réalisation de ce TB. Un *repository* associé à ce travail a été déployé sur *Github*.

LaTex

Le site http://www.tablesgenerator.com/latex_tables a été utilisé afin de pouvoir générer facilement des tables *LaTex*.

1. *Visual Studio Code* : <https://code.visualstudio.com/>

Authentification

Je confirme avoir réalisé seul ce travail et ne pas avoir utilisé d'autre sources que celles citées dans la biographie.

Date

Rémi Jacquemard

Bibliographie

- ALMEIDA, P. et al. *Parking Lot Database - Laboratório Visão Robótica e Imagens*. URL : <https://web.inf.ufpr.br/vri/databases/parking-lot-database/> (visité le 18/05/2018).
- « PKLot – A robust dataset for parking lot classification ». In : *Expert Systems with Applications* 42 (2015), p. 4937-4949.
- AMATO, Giuseppe et al. « Deep Learning for Decentralized Parking Lot Occupancy Detection ». In : *Expert Syst. Appl.* 72.C (avr. 2017), p. 327-334. ISSN : 0957-4174. DOI : 10.1016/j.eswa.2016.10.055. URL : <https://doi.org/10.1016/j.eswa.2016.10.055>.
- ELECTROSUN. *Caméra IP solaire autonome*. URL : <http://electrosun.fr/Cam%C3%A9ra-surveillance-solaire> (visité le 18/03/2018).
- FILL. *Parking automobiles véhicules*. 13 juin 2015. URL : <https://pixabay.com/fr/parking-automobiles-v%C3%A9hicules-825371/> (visité le 18/05/2018).
- FLASK. *Flask (A Python Microframework)*. URL : <http://flask.pocoo.org/> (visité le 21/07/2018).
- GEEKS, Les. *[TEST] Présentation De La Caméra IP SOLAIRE HD 720p WANSCAM HW0029*. 17 déc. 2016. URL : <https://lesbonstuyauxgeeks.fr/test-presentation-de-la-camera-ip-solaire-hd-720p-wanscam-hw0029/> (visité le 11/03/2018).
- KERAS. *Keras: The Python Deep Learning library*. URL : <https://keras.io/> (visité le 22/07/2018).
- *Tensorflow Object Detection API*. URL : https://github.com/tensorflow/models/tree/master/research/object_detection (visité le 18/06/2018).
- KRAUSE, Jonathan et al. « 3D Object Representations for Fine-Grained Categorization ». In : *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.
- LI, Fei-Fei, Andrej KARPATHY et Justin JOHNSON. *Parking automobiles véhicules*. 1^{er} fév. 2016. URL : <https://web.cs.hacettepe.edu/~aykut/classes/spring2016/bil722/slides/w05-FasterR-CNN.pdf> (visité le 19/07/2018).
- MAGICVISION. *Aliexpress.com – Wanscam HW0029-5 Extérieure Étanche*. URL : <https://fr.aliexpress.com/item/Wanscam-HW0029-3-720P-solar-wifi-IP-camera-build-in-16G-TF-cards-support-128G-TF/32795681508.html> (visité le 13/03/2018).
- MAPS, Google. *HEIG-VD*. 2018. URL : <https://www.google.com/maps/@46.7794439,6.6587222,18z> (visité le 15/04/2018).
- MATLAB. *MATLAB - MathWorks*. URL : <https://www.mathworks.com/products/matlab.html> (visité le 14/06/2018).
- MCDONALD, Donor. *An Intuitive Explanation of Convolutional Neural Networks*. 11 août 2016. URL : <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/> (visité le 17/07/2018).
- *Machine learning fundamentals (II): Neural networks*. 21 déc. 2017. URL : <https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef> (visité le 17/07/2018).
- MSCOCO. *Coco: Common Object in Context*. URL : <cocodataset.org/> (visité le 22/07/2018).

- NG, Andrew, Kian KATANFOROOSH et Younes BENSOUDA MOURRI. *Object detection*. 2018. URL : <https://www.coursera.org/lecture/convolutional-neural-networks/object-localization-nEeJM> (visité le 27/06/2018).
- NUMPY.ORG. *NumPy*. URL : <http://www.numpy.org/> (visité le 14/06/2018).
- PEARL.CH. *Caméra HD-IP*. URL : <https://fr.pearl.ch/article/NX4377/camera-hd-ip-avec-batterie-rechargeable-et-panneau-solaire-ipc-790-solar> (visité le 09/03/2018).
- PJREDDIE. *YOLO: Real-Time Object Detection*. URL : <https://pjreddie.com/darknet/yolo/> (visité le 26/06/2018).
- PYTHON.ORG. *Logging facility for Python*. URL : <https://docs.python.org/3/library/logging.html> (visité le 14/06/2018).
- *Welcome to Python.org*. URL : <https://www.python.org/> (visité le 14/06/2018).
- R-PROJECT. *The R Project for Statistical Computing*. URL : <https://www.r-project.org/> (visité le 14/06/2018).
- REDMON, Joseph et Ali FARHADI. « YOLOv3: An Incremental Improvement ». In : *arXiv* (2018).
- REN, Shaoqing et al. « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ». In : *CoRR* abs/1506.01497 (2015). arXiv : 1506.01497. URL : <http://arxiv.org/abs/1506.01497>.
- REOLINK. *Argus 2 - Wire-Free Camera*. 2018. URL : <https://reolink.com/product/argus-2> (visité le 13/03/2018).
- SCIKIT-IMAGE.ORG. *Scikit-image: Image processing in Python*. URL : scikit-image.org (visité le 14/06/2018).
- STORE, Damien ABAD - Domotique. *API des CAMÉRAS IP Wanscam Onvif*. 29 avr. 2016. URL : <http://tutoriels.domotique-store.fr/content/52/293/fr/api-des-cameras-ip-wanscam-onvif-hw-nouvelle-generation.html> (visité le 11/03/2018).
- TEAM, OpenCV. *OpenCV library*. URL : <https://opencv.org/> (visité le 14/06/2018).
- TENSORFLOW. *TensorFlow*. URL : <https://www.tensorflow.org/> (visité le 19/07/2018).
- VALIPOUR, Sepehr et al. « Parking Stall Vacancy Indicator System Based on Deep Convolutional Neural Networks ». In : *CoRR* abs/1606.09367 (2016). arXiv : 1606.09367. URL : <http://arxiv.org/abs/1606.09367>.
- VIDEOSURVEILLANCE-BOUTIQUE.FR. *Comment bien choisir sa caméra de surveillance*. URL : <http://www.videosurveillance-boutique.fr/support/comment-bien-choisir-sa-camera-de-surveillance-483.html> (visité le 12/03/2018).
- WANSACM. *HW0029-3 - Outdoor IP Camera*. URL : <http://wanscam.com/productshow-14-65-1.html> (visité le 11/03/2018).
- *HW0029-5 - Outdoor IP Camera 1080P*. URL : <http://wanscam.com/productshow-16-94-1.html> (visité le 12/03/2018).
- WIKIPEDIA. *Basic access authentication*. 7 juin 2018. URL : https://en.wikipedia.org/wiki/Basic_access_authentication (visité le 13/06/2018).
- *Fully qualified domain name*. 25 déc. 2016. URL : https://fr.wikipedia.org/wiki/Fully_qualified_domain_name (visité le 13/06/2018).

- WIKIPEDIA. *HSL and HSV*. 28 juin 2018. URL : https://en.wikipedia.org/wiki/HSL_and_HSV (visité le 09/07/2018).
- *Inpainting*. URL : <https://en.wikipedia.org/wiki/Inpainting> (visité le 24/07/2018).
- *Machine Learning*. 29 juin 2018. URL : https://en.wikipedia.org/wiki/Machine_learning (visité le 17/07/2018).
- *ONVIF*. 18 fév. 2018. URL : <https://en.wikipedia.org/wiki/ONVIF> (visité le 07/03/2018).
- *Real Time Streaming Protocol*. 27 jan. 2018. URL : https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol (visité le 07/03/2018).
- *Régression (statistique)*. 24 oct. 2017. URL : [https://fr.wikipedia.org/wiki/R%C3%A9gression_\(statistiques\)](https://fr.wikipedia.org/wiki/R%C3%A9gression_(statistiques)) (visité le 19/07/2018).
- *Réseau de neurones artificiels*. 13 juil. 2018. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels (visité le 17/07/2018).
- *Réseaux neuronal convolutif*. 5 juin 2018. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif (visité le 19/07/2018).
- *Root Mean Squared Deviation*. URL : https://en.wikipedia.org/wiki/Root-mean-square_deviation (visité le 23/07/2018).
- *Tenseur*. 7 juin 2018. URL : <https://fr.wikipedia.org/wiki/Tenseur> (visité le 19/07/2018).
- *TensorFlow*. 28 juin 2018. URL : <https://en.wikipedia.org/wiki/TensorFlow> (visité le 22/07/2018).
- WU, Tianfu, Bo LI et Song-Chun ZHU. « Learning And-Or Models to Represent Context and Occlusion for Car Detection and Viewpoint Estimation ». In : *CoRR* abs/1501.07359 (2015). arXiv : 1501.07359. URL : <http://arxiv.org/abs/1501.07359>.
- ZONEBOURSE.COM. *EURO / SWISS FRANC (EUR/CHF)*. URL : <https://www.zonebourse.com/EURO-SWISS-FRANC-EUR-C-4594/graphiques-cours/> (visité le 12/05/2018).

Table des figures

2.1	Exemple d'images du corpus utilisé	11
3.1	Image de faible définition dont les pixels sont apparents	13
3.2	Représentation d'une image couleur 4x4 pixels et les composantes rouges, vertes, et bleues	14
3.3	Cône représentant l'espace HSV. Ici, la saturation est désignée par le terme <i>Chroma</i> .	14
3.4	Image de parking avant détection de bord	15
3.5	Image de parking après détection de bord	15
3.6	Exemple de réseau de neurones	16
3.7	Exemple de réseau de neurones à convolutions	17
3.8	Architecture <i>Faster R-CNN</i>	18
3.9	Architecture <i>Yolo</i>	19
4.1	Capture d'images de plusieurs caméras	20
4.2	Caméra alimentée et connectée au réseau par câble	21
4.3	Caméra câblée au réseau et alimentée par <i>PoE</i>	21
4.4	Caméra alimentée par câble et connectée au réseau en Wifi	22
4.5	Caméra auto-alimentée et connectée en Wifi	22
4.6	Electrosun Caméra de surveillance solaire	27
4.7	Reolink Argus 2	27
4.8	Wanscam HW0029-3	28
4.9	Wanscam HW0029-5	30
4.10	Caméra Visortech	32
4.11	Images d'exemple provenant du dataset <i>Cars</i>	37
4.12	Image d'exemple provenant du dataset <i>PKLot</i>	38
4.13	Image d'exemple provenant du dataset <i>PKLot</i> qui sera utilisée afin de définir les divers paramètres	39
4.14	Image originale rééchantillonnée à 320 x 180	40
4.15	L'image originale avec différentes détections de bords	41
4.16	Comparaison des deux méthodes de traitements	42
4.17	Image traitée, capturée depuis la caméra placée sur le toit de la HEIG-VD	43
4.18	Suppression d'arrière plan	44
4.19	Proposition de modèle de réseau de neurones à convolutions pour résoudre le problème de régression	46
4.20	Image de parking subdivisée selon une grille	48
4.21	Proposition de modèle de réseau de neurones à convolutions, avec une grille en sortie .	49
4.22	Capture d'images de plusieurs caméras	51
5.1	Emplacement de la caméra (en rouge) et du parking filmé (en vert)	54
5.2	Requête d'une image à la caméra Wanscam <i>HW0029-5</i>	55
5.3	Requêtes d'images et traitement	58
5.4	Avant-après traitements d'une image du corpus <i>PKLot</i>	63
6.1	Entrainement sur le corpus <i>PKLot</i> (graphes par <i>Tensorboard</i>)	64

6.2	Exemple de détection sur un entraînement de 4000 itérations	65
6.3	Exemple de détection avec un modèle <i>Yolo v3</i> pré-entraîné sur <i>Coco</i>	66
A.1	L'image originale redimensionnée à différentes résolutions	72
A.2	<i>Scharr HSV</i> : image utilisée afin d'explorer les différents sous-échantillonnages	73
A.3	L'image dont les bords ont été détectés, redimensionnée à différentes résolutions	74
A.4	480 x 270 : image utilisée afin d'explorer les différentes méthodes de détection de bords	75
A.5	L'image sous-échantillonnée avec différentes détections de bords	76