

AURÉLIEN LEBRUN

RAPPORT DE PFE

DÉTECTION DE PLANS ET D'OBJETS DANS
UNE IMAGE PAR MACHINE LEARNING

TUTEUR ENTREPRISE : ERIC REMILLERET

TUTEUR ÉCOLE : LUC JAULIN

8 septembre 2020



Remerciements

Avant de commencer ce rapport, je souhaite remercier les personnes et les organisations qui ont permis que ce stage se déroule dans les meilleures conditions.

En premier lieu, je souhaite remercier tout particulièrement Eric REMILLERET, mon tuteur de stage, pour son écoute, sa confiance et son suivi tout au long de cette aventure. Je remercie aussi Arjen KRANEVELD, Didier GIOT et Morgan BUISSON pour leurs aides et leurs conseils. Je remercie toute mon équipe projet, Arthur KAPLAN, Thomas ARNAUD, Julien BEYER, Tony CHAN et Elouan COURTEL, pour leur travail, leur implication et leur bonne humeur.

Je remercie aussi l'entreprise AUBAY qui m'a permis de rentrer dans la vie professionnelle. Leur suivi et leur aide m'ont permis de progresser et d'effectuer ce stage dans de très bonnes conditions malgré les évènements sanitaires actuels.

Je tiens aussi à remercier mon professeur référent et professeur principal de spécialité robotique, Luc JAULIN. Son enseignement et ses qualités pédagogiques m'ont fait énormément progresser et m'ont poussé à choisir un stage orienté recherche.

Finalement, je remercie l'ENSTA Bretagne pour le suivi et les compétences que cette école m'a apportés.

Résumé

Ce rapport présente les travaux réalisés dans le cadre du projet de fin d'études avec l'entreprise AUBAY. Ce stage a été effectué sous la tutelle d'Eric REMILLERET et Luc JAULIN. Il fait suite à trois années d'études à l'ENSTA Bretagne. Son objectif est de créer un algorithme capable de détecter la profondeur et les objets présents dans une image. Le choix de ce projet a été motivé par sa dimension innovante ainsi que par l'opportunité de travailler sur un sujet traitant d'intelligence artificielle.

Pour la partie estimation de profondeur dans l'image, le travail réalisé consiste en l'analyse précise et l'amélioration de deux réseaux de neurones. Pour la partie détection d'objets, il consiste en l'utilisation de plusieurs réseaux de neurones et la mise en place d'un algorithme complet de traitement d'images pour améliorer les résultats des algorithmes de Machine Learning.

Ce projet innovant a débuté cette année et a répondu aux exigences du sujet. En effet, un algorithme capable de décrire efficacement l'image en termes de profondeur et d'objets la composant a été créé. Une poursuite du travail est nécessaire pour améliorer l'algorithme actuel et passer outre les conditions définies sur les images d'entrée.

Abstract

This report presents the work carried out in the context of an end of studies internship with the AUBAY company. This internship was pursued under the supervision of Eric REMILLE-RET and Luc JAULIN. It concludes three years of intensive studies at ENSTA Bretagne. Its purpose is to create an algorithm capable of detecting depth and objects within an image. The choice of this subject was motivated by the innovative aspect and by the opportunity to work on a subject dealing with artificial intelligence.

The first part of the project focused on the depth estimation, the work consisted of a precise analysis and trying to improve two neural networks. For the second part dealing with object detection, the task at hand consisted in the use of many neural networks and the development of a complete and complex image processing algorithm to improve neural networks results.

This innovative project started this year and met the expectations of the subject. Indeed, an algorithm able to efficiently describe an image in terms of depth and objects was created. Further work is now necessary to improve the actual algorithm and bypass its current limitations concerning input type of images.

Table des matières

| | |
|--|-----------|
| Résumé | 2 |
| Abstract | 3 |
| Introduction | 6 |
| 1 Contexte | 7 |
| 1.1 Présentation de l'entreprise | 7 |
| 1.2 Présentation du projet | 8 |
| 1.3 Organisation | 8 |
| 1.3.1 Période de télétravail | 8 |
| 1.3.2 Différentes phases | 8 |
| 1.3.3 Méthodologie Agile | 9 |
| 2 Préliminaires | 11 |
| 2.1 Plusieurs approches du sujet | 11 |
| 2.1.1 Approche physique | 11 |
| 2.1.2 Approche par Machine Learning | 11 |
| 2.1.2.1 Notions importantes | 11 |
| 2.1.2.2 Architecture DenseDepth | 12 |
| 2.1.2.3 Architecture RSDE | 13 |
| 2.2 Choix et limitations | 14 |
| 2.2.1 Réseaux de neurones | 14 |
| 2.2.2 Datasets | 15 |
| 3 Contribution | 17 |
| 3.1 Structure globale | 17 |
| 3.2 Estimation de profondeur | 19 |
| 3.2.1 Carte des erreurs | 19 |
| 3.2.2 Découpage de l'image en plans | 20 |
| 3.2.3 Généralisation du modèle | 23 |
| 3.2.4 Nouvelle architecture | 24 |
| 3.3 Segmentation d'image | 25 |
| 3.3.1 Choix | 25 |
| 3.3.2 Amélioration de la segmentation | 29 |
| 3.4 Optimisation de l'estimation de profondeur | 34 |
| 3.4.1 Principe | 34 |
| 3.4.2 Résultats | 35 |

| | |
|--------------------------------------|-----------|
| 4 Application | 36 |
| 4.1 Présentation | 36 |
| 4.2 Interface graphique | 36 |
| 4.3 Fonctionnalités | 38 |
| | |
| Conclusion | 43 |
| | |
| Annexes | 43 |
| | |
| A Glossaire | 44 |
| | |
| B Filtre de Sobel | 46 |
| | |
| C Couche des réseaux DenseNet | 48 |

Introduction

La connaissance précise de l'environnement qui entoure une machine est un enjeu majeur actuellement. La course à la création de voitures autonomes le montre bien [17, 19]. En effet, sans l'information de ce qui se trouve autour d'elle et de leur distance précise, une telle voiture ne peut pas se déplacer sans risque.

Les technologies utilisées pour obtenir l'information de profondeur sont dans ce cas souvent des caméras stéréoscopiques ou des lidars [3]. L'entreprise AUBAY a voulu aller plus loin et agrandir les champs d'application en créant des algorithmes capables d'identifier précisément l'environnement en utilisant uniquement une image simple. C'est dans ce contexte que l'entreprise a proposé le sujet ambitieux sur lequel porte ce rapport.

AUBAY est une entreprise française des services du numérique et l'une des rares à avoir une cellule recherche. C'est dans une équipe de recherche que s'est déroulé ce stage, la mission consistant dans le développement de l'application, mais aussi dans la transmission du savoir acquis pendant six mois.

Le premier chapitre présentera le contexte de déroulement du stage. Le deuxième chapitre discutera des choix et restrictions faits sur le sujet. Le troisième chapitre détaillera les travaux réalisés sur les différentes parties du projet. Finalement, le dernier chapitre présentera l'application finale réalisée au terme de ce stage.

Chapitre 1

Contexte

1.1 Présentation de l'entreprise

Présentation générale

AUBAY est une Entreprise de Services de Numérique composée d'environ 6500 collaborateurs, répartis dans sept pays européens. Elle se positionne sur sept secteurs d'activité mais réalise la plus grande partie de son chiffre d'affaires dans les secteurs des banques, assurances et télécoms. AUBAY est répartie en plusieurs *Business Units* comme : "Assurance", "Banque", "Innovation", "Finance", "Telecom, Média, Jeux".

Ce stage s'est déroulé dans la plus grande entité du groupe, située à Boulogne-Billancourt, dans la cellule Innovation.

Cellule Innovation

Cette cellule est la cellule orientée recherche du groupe. Elle réalise des projets variés sur des sujets innovants. Sa raison d'exister est à la fois de pouvoir anticiper des demandes clients mais aussi de pouvoir agir comme image de marque de l'entreprise. Elle est une vitrine sur les capacités d'innovation d'AUBAY.

Plusieurs projets sont actuellement en développement :

- **WMF** : L'objectif est de créer un algorithme permettant de reconnaître les émotions humaines.
- **VR CLOUD** : L'objectif est de représenter l'architecture Cloud d'AUBAY à l'aide de la réalité virtuelle.
- **EPCL** : L'objectif est de proposer un outil web de visualisation et d'analyse de parcours client.
- **ISCAN** : L'objectif est de créer un algorithme de reconnaissance d'écriture.
- **GO CHAIN** : L'objectif est d'utiliser la blockchain pour réaliser des applications cryptographiques.

- **AMC** : L'objectif est de créer un robot capable de se déplacer dans un environnement inconnu.
- **KYP** : Le projet du stage, dont l'objectif sera détaillé dans la section suivante.
- **DWYH** : L'objectif est de pouvoir contrôler un ordinateur à l'aide d'une webcam.
- **SERIOUS** : L'objectif est de créer un jeu permettant de simuler la journée d'un salarié AUBAY.
- **NLP** : L'objectif est de créer une application permettant de résumer les réunions automatiquement.
- **AMP** : L'objectif est de créer une application capable de créer de la musique.

1.2 Présentation du projet

Le projet Know Your Picture a pour objectif d'extraire le plus précisément possible les informations d'une image, les axes principaux de recherche étant l'estimation de la profondeur et la détection des objets. La contrainte principale du projet est le format des images utilisées. En effet, la profondeur d'une image est très souvent estimée grâce à des images stéréoscopiques qui permettent d'obtenir des informations sur la profondeur beaucoup plus facilement grâce à des concepts de traitement d'image. Ici, les images utilisées ne seront pas stéréoscopiques, ce seront des images simples.

Ce projet a débuté en 2020 avec l'équipe actuelle. Elle était composée de six membres, cinq en stage de fin d'études et un en stage d'avant dernière année.

1.3 Organisation

1.3.1 Période de télétravail

Ce stage a naturellement été impacté par cette période particulière de crise sanitaire. Différentes mesures ont été mises en place.

Durant le confinement, le travail a été effectué en télétravail. Le suivi par les encadrants d'AUBAY s'est poursuivi en gardant le même nombre de réunions mais en changeant leur forme puisque celles-ci se sont déroulées sur l'application Teams. Les encadrants sont cependant restés à l'écoute et disponibles quelles que soient les interrogations survenues durant ce projet.

Après la période de confinement, une période de télétravail partiel s'est mise en place. Ainsi, la semaine était découpée en deux jours sur site et trois jours en télétravail. Le stage s'est terminé sur ce format.

1.3.2 Différentes phases

Le stage s'est déroulé en trois phases : la phase d'état de l'art, la phase de preuve de concept et la phase de projet.

La phase d'état de l'art consiste en la recherche de l'existant. Pour chaque méthode trouvée, un rapport la résumant et précisant les points clés ainsi que les premiers ressentis est écrit.

L'objectif est de permettre à tous les membres du groupe de pouvoir discuter ensemble sur les méthodes à garder pour la phase de preuve de concept, grâce à la lecture de ces rapports. Cela permet aussi de discuter des différentes méthodes avec les encadrants.

La phase de preuve de concept est la phase la plus longue du stage. Une fois la ou les méthodes choisies grâce à la phase précédente, les membres du groupe se répartissent les tâches pour essayer de les analyser, les mettre en place et les améliorer. Dans le cas de l'équipe KYP, le projet débutait et énormément de méthodes étaient disponibles. Il a donc été décidé d'en choisir une par personne parmi les plus prometteuses pour travailler dessus.

La dernière phase est la phase de projet. Celle-ci est plus courte et a pour objectif de mettre en avant les travaux réalisés pendant la deuxième phase en créant une application. Cette phase a duré un mois. Elle donne lieu à une présentation devant la direction d'AUBAY et toute personne de l'entreprise intéressée. Cette année, cette présentation a été effectuée à travers Teams devant plus de soixante personnes.

1.3.3 Méthodologie Agile

Tout le stage s'est déroulé sous une organisation dite "Agile". En effet, le travail s'est articulé autour de "*sprints*" d'une semaine. C'est-à-dire que chaque semaine, un compte-rendu était fait sur l'avancée des travaux et un nouvel objectif était fixé en fonction de celle-ci. Deux réunions ont été organisées par semaine, un comité de suivi et un "*stand-up*". Le comité de suivi est une réunion longue où chaque point de blocage est abordé en détails et où les nouveaux objectifs sont fixés. Le "*stand-up*" est une réunion courte où seuls les points de blocage critiques sont discutés. Cette dernière se tient normalement debout, d'où le nom de "*stand-up*". Ces réunions régulières permettent de s'adapter aux différents problèmes et de favoriser l'échange entre les membres du groupe ainsi qu'avec les encadrants du projet.

La communication entre collègues était assurée grâce à l'application Teams. Pour la communication sur les différentes tâches, le site agile "Jira" a été utilisé. Il permet de créer des projets dans lesquels des tickets sont créés et manipulés. Chaque ticket correspond à une tâche et sa position dans le tableau, dont un exemple est présenté dans la figure 1.1, permet de connaître son état actuel. Ainsi, on peut en un coup d'œil connaître les tâches choisies pour le développement, celles en cours de développement, celles à valider et celles finies et validées. L'utilisation d'un tel outil a permis de répartir efficacement les tâches, sans effectuer de doublons.

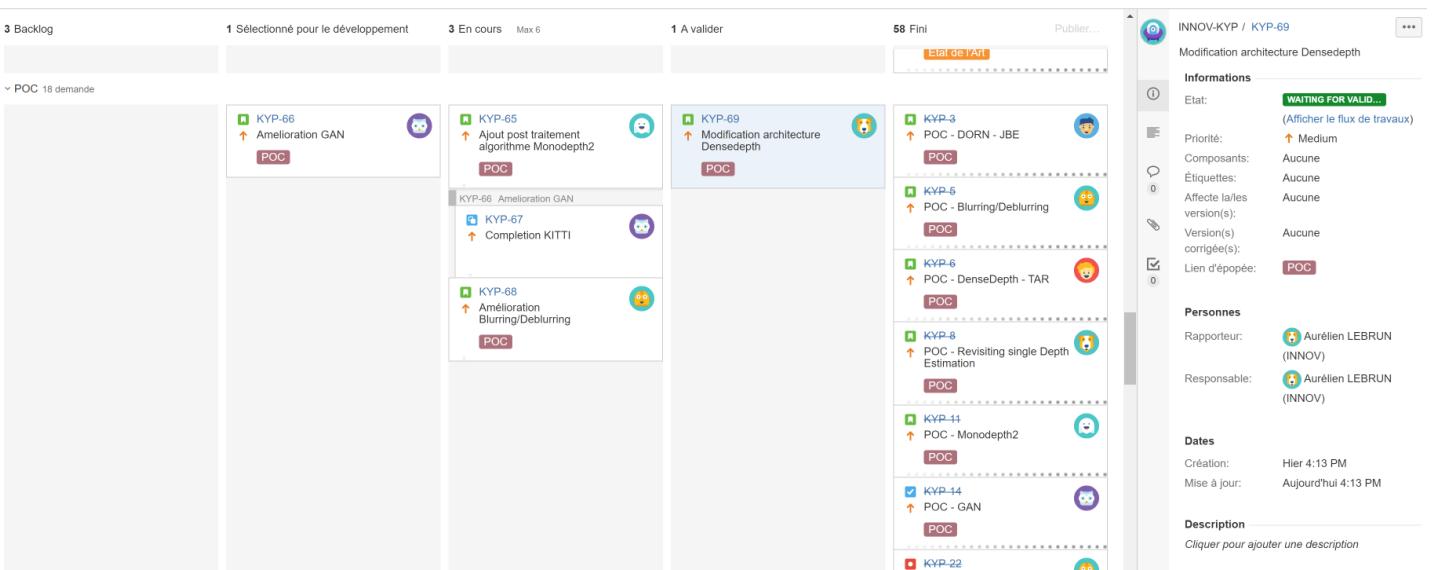


FIGURE 1.1 – Tableau agile sur le site Jira

Chapitre 2

Préliminaires

2.1 Plusieurs approches du sujet

La partie état de l'art du projet a amené à isoler deux approches possibles du sujet. La première est une approche par *Machine Learning*. Elle a l'avantage d'offrir des résultats visuels prometteurs. Cependant, ses résultats, comme c'est souvent le cas en *Machine Learning*, sont très liés aux images avec lesquelles on entraîne le modèle. La deuxième approche est une méthode qui se base sur des considérations purement physiques. Elle a l'avantage de ne pas nécessiter d'entraînement et d'être théoriquement utilisable sur n'importe quel type d'image. Elle a cependant le désavantage de présenter des résultats moins prometteurs. Ces deux approches ont été explorées par l'équipe KYP. L'approche physique ayant été travaillée par un autre membre de l'équipe, sa présentation en sera succincte. Les deux modèles de *Machine Learning* étudiés seront présentés à la fin de la section.

2.1.1 Approche physique

Cette méthode part du postulat que chaque photographie comporte des parties nettes, là où est faite la mise au point, et d'autres qui ne le sont pas, devant et derrière le plan de focus. Il existe donc un lien entre la profondeur d'un pixel et son niveau de flou.

La méthode développée est composée de trois étapes. Une première étape de *Blurring* qui sert à déterminer la quantité de flou qu'il faudrait ajouter à un pixel pour qu'il semble être à l'infini. La seconde étape est une étape de *Deblurring* qui permet de déterminer la quantité de flou à enlever à un pixel pour qu'il semble être dans le plan de focus. La dernière étape est une étape d'optimisation permettant de faire coïncider les deux mesures précédentes [23].

2.1.2 Approche par Machine Learning

2.1.2.1 Notions importantes

Les modèles utilisés possèdent une structure de type encodeur-décodeur ainsi que du *transfer learning*. Ces deux notions seront expliquées dans un premier temps.

Encodeur-décodeur

Un modèle de type encodeur-décodeur [28] possède une architecture particulière. Celle-ci est composée de deux parties, une partie encodeur et une partie décodeur. La partie encodeur prend l'entrée du modèle et la transforme en vecteurs représentant des informations sur celle-ci. Dans le cas d'une image, cette partie a pour effet de réduire sa taille. La deuxième partie, le décodeur, prend ces vecteurs en entrée pour les transformer en la sortie voulue. Dans le cas d'une image, cela va agrandir la taille des vecteurs pour les retransformer en une image d'une taille convenable. Pour réaliser cet agrandissement, des informations de l'encodeur, lorsque l'image était plus grande, sont réutilisées dans le décodeur. Cela est possible grâce à un procédé dit de *skip connections* [25] qui crée un lien entre deux couches du réseau. Dans la figure 2.1, on observe que la taille de l'image initiale se réduit, due notamment à des opérations de *pooling* [16]. Inversement, dans le décodeur, des opérations d'agrandissement permettent d'augmenter la taille de l'image. Grâce aux flèches sur la figure, on observe bien que les opérations d'agrandissement utilisent les informations de l'encodeur. Théoriquement, chacune de ces parties peuvent fonctionner indépendamment. Cependant, les très bons résultats des encodeurs-décodeurs justifient leur utilisation.

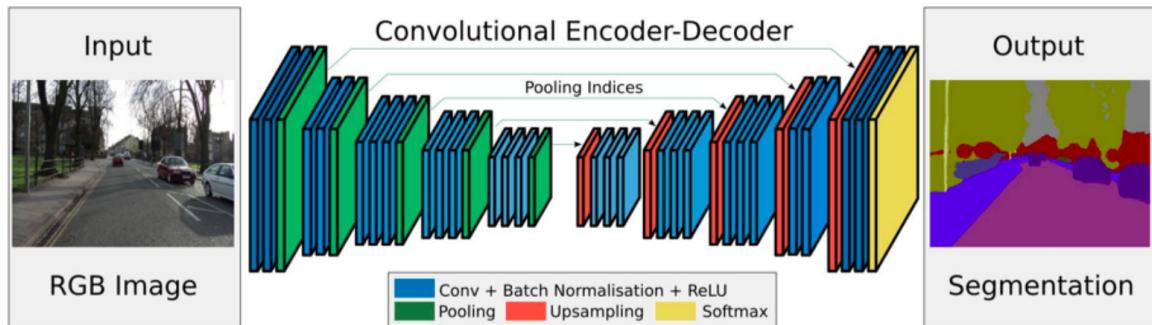


FIGURE 2.1 – Schéma d'une architecture de type encodeur-décodeur

Transfer Learning

Le *transfer learning* [6] est une technique très utilisée en *Machine Learning* en raison de l'énorme puissance de calcul nécessaire à l'entraînement des modèles. Cette technique consiste à réentraîner un modèle déjà entraîné sur un autre jeu de données. On peut choisir de réentraîner tous les neurones du réseau ou seulement une partie. Cela permet d'obtenir de meilleurs résultats et de réutiliser des calculs déjà effectués préalablement. L'idée est de se dire qu'un réseau déjà entraîné à reconnaître une classe ayant des ressemblances avec la classe que l'on veut détecter est un bon point de départ pour commencer l'entraînement du modèle. Par exemple, un réseau capable de détecter des vélos sera un bon point de départ si on veut créer un modèle capable de détecter des motos.

2.1.2.2 Architecture DenseDepth

Ce premier modèle est de type encodeur-décodeur. L'encodeur est un réseau "DenseNet 169" [11], préentraîné sur le dataset ImageNet par *transfer learning*. Il est composé de couches

particulières appelées "couches denses". Chacune de ces couches présente la particularité d'être connectée à toutes les suivantes. La description précise des couches de ce réseau est présente dans l'annexe C. Le décodeur est composé de plusieurs couches d'agrandissement et de convolution. Des *skip connections* sont utilisées dans le décodeur afin d'avoir accès à des informations de l'encodeur. La description précise de l'architecture est présente dans la figure 2.2.

| LAYER | OUTPUT | FUNCTION |
|------------------|-----------------------------|--|
| INPUT | $480 \times 640 \times 3$ | |
| CONV1 | $240 \times 320 \times 64$ | DenseNet CONV1 |
| POOL1 | $120 \times 160 \times 64$ | DenseNet POOL1 |
| POOL2 | $60 \times 80 \times 128$ | DenseNet POOL2 |
| POOL3 | $30 \times 40 \times 256$ | DenseNet POOL3 |
| ... | ... | ... |
| CONV2 | $15 \times 20 \times 1664$ | Convolution 1×1 of DenseNet BLOCK4 |
| UP1 | $30 \times 40 \times 1664$ | Upsample 2×2 |
| CONCAT1 | $30 \times 40 \times 1920$ | Concatenate POOL3 |
| UP1-CONVA | $30 \times 40 \times 832$ | Convolution 3×3 |
| UP1-CONVB | $30 \times 40 \times 832$ | Convolution 3×3 |
| UP2 | $60 \times 80 \times 832$ | Upsample 2×2 |
| CONCAT2 | $60 \times 80 \times 960$ | Concatenate POOL2 |
| UP2-CONVA | $60 \times 80 \times 416$ | Convolution 3×3 |
| UP2-CONVB | $60 \times 80 \times 416$ | Convolution 3×3 |
| UP3 | $120 \times 160 \times 416$ | Upsample 2×2 |
| CONCAT3 | $120 \times 160 \times 480$ | Concatenate POOL1 |
| UP3-CONVA | $120 \times 160 \times 208$ | Convolution 3×3 |
| UP3-CONVB | $120 \times 160 \times 208$ | Convolution 3×3 |
| UP4 | $240 \times 320 \times 208$ | Upsample 2×2 |
| CONCAT3 | $240 \times 320 \times 272$ | Concatenate CONV1 |
| UP2-CONVA | $240 \times 320 \times 104$ | Convolution 3×3 |
| UP2-CONVB | $240 \times 320 \times 104$ | Convolution 3×3 |
| CONV3 | $240 \times 320 \times 1$ | Convolution 3×3 |

FIGURE 2.2 – Couches du modèle DenseDepth. Les couches jusqu'à CONV2 sont exactement celles du réseau DenseNet169. Chaque couche CONVB est une couche de convolution suivie d'une activation ReLU [18]

2.1.2.3 Architecture RSDE

L'architecture RSDE [10] est plus complexe que l'architecture précédente. Elle est composée de quatre modules. Premièrement, un module encodeur extrait des informations de l'image aux échelles $1/4$, $1/8$, $1/16$, $1/32$. L'idée de cette extraction à plusieurs échelles est de garder en mémoire toutes les informations de l'encodeur et pas uniquement l'extraction finale. En effet, les premières couches de l'encodeur vont permettre d'extraire des caractéristiques générales sur l'image tandis que les dernières couches vont permettre d'extraire des caractéristiques plus précises sur celle-ci. Le deuxième module est un module décodeur qui utilise quatre modules d'agrandissement pour augmenter la taille de la sortie de l'encodeur tout en réduisant le nombre de canaux dans l'image. L'encodeur et le décodeur utilisent une architecture SENet-154. Le

troisième module est un module utilisé pour fusionner les informations extraites par l'encodeur. Chaque vecteur ainsi extrait est redimensionné à la taille de l'image de sortie grâce à quatre modules d agrandissement. Ces images sont ensuite concaténées. Finalement, le dernier module permet de concaténer les résultats du décodeur et du troisième module. Il effectue finalement trois couches de convolutions avant de ressortir le résultat final du modèle.

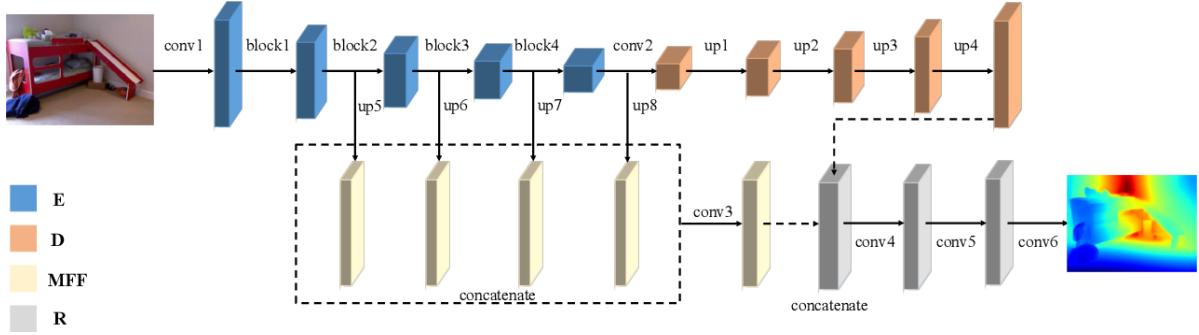


FIGURE 2.3 – Architecture du modèle RSDE

2.2 Choix et limitations

2.2.1 Réseaux de neurones

Lors de ce stage, plusieurs choix importants se sont présentés. Le premier fut de choisir un réseau de neurones sur lequel travailler. Les recherches durant la partie d'état de l'art ont permis d'isoler plusieurs modèles. Le tableau 2.1 résume leurs performances en se basant sur les métriques suivantes :

$$\begin{aligned} \text{average relative error (REL)} &= \frac{1}{n} \sum_p^n \frac{|y_p - \hat{y}_p|}{y}, \\ \text{root mean squared error (RMS)} &= \sqrt{\frac{1}{n} \sum_p^n (y_p - \hat{y}_p)^2}, \\ \text{average } (\log_{10}) \text{ error} &= \frac{1}{n} \sum_p^n |\log_{10}(y_p) - \log_{10}(\hat{y}_p)|. \end{aligned}$$

avec y_p un pixel dans l'image de profondeur de référence y , \hat{y}_p un pixel dans l'image de profondeur prédictive \hat{y}_p , n le nombre de pixels total dans chaque image de profondeur.

| Method | REL | RMS | \log_{10} |
|-------------------|-------|-------|-------------|
| Eigen et al. [4] | 0.158 | 0.641 | . |
| Laina et al. [15] | 0.127 | 0.573 | 0.055 |
| MS-CRF [27] | 0.121 | 0.586 | 0.052 |
| Hao et al. [7] | 0.127 | 0.555 | 0.053 |
| Fu et al. [5] | 0.115 | 0.509 | 0.051 |
| DenseDepth [1] | 0.123 | 0.465 | 0.053 |
| RSDE [10] | 0.115 | 0.530 | 0.050 |

TABLE 2.1 – Métriques des différents réseaux de neurones

Si l'on regarde la métrique RMS qui est souvent utilisée comme référence, on observe qu'un seul modèle passe en dessous de la barre des 0.5. Cette métrique est normée entre 0 et 1 et plus elle est proche de 0, meilleure elle est. On peut alors conclure que les résultats de ces modèles ne sont pas bons. Cependant, ce sont les meilleurs développés actuellement. Le deuxième meilleur modèle selon ces métriques, le RSDE, a été choisi pour l'analyse. Ce choix a été effectué car un autre membre du groupe travaillait déjà sur l'analyse du meilleur modèle, le DenseDepth. Un travail a cependant été réalisé sur ce dernier dans le but de l'améliorer en changeant son architecture, comme cela sera présenté dans la section 3.2.4.

2.2.2 Datasets

Un autre choix important fut les *datasets* avec lesquels travailler. Ceux utilisés sont très particuliers et il était impossible d'en créer un totalement nouveau. En effet, chaque élément du *dataset* doit être constitué de deux éléments : une image et sa carte de profondeur. Une carte de profondeur est une image qui associe à chaque pixel la valeur de profondeur de l'image qu'elle représente. Des outils spécifiques sont nécessaires pour créer une telle image. Peu de *datasets* de ce type existent actuellement.

Deux *datasets* ont été choisis :

- *Dataset NYU V2* : Ce dataset est composé de 407 024 images de profondeur ainsi que de 1449 images de profondeurs labélisées pour de la segmentation sémantique instanciée. Ces images sont obtenues à l'aide d'une kinect. Les images sont disponibles dans leur version "non traitée" ainsi que dans une version "traitée" qui comble les trous présents dans les images de profondeur. Les images sont prises sur une variété de 464 scènes d'intérieur différentes.

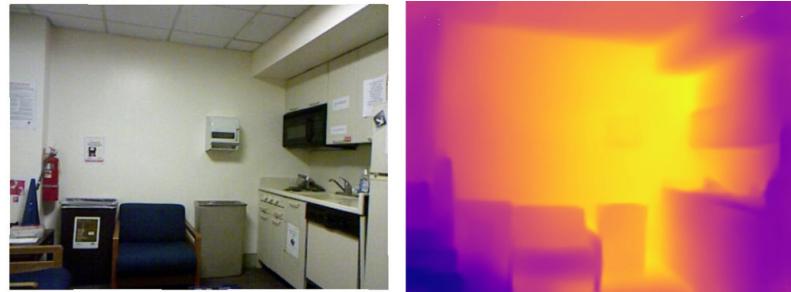


FIGURE 2.4 – Image du dataset NYU V2 (à gauche) avec la carte de profondeur correspondante (à droite)

- *Dataset KITTI* : Plus précisément, la partie "depth" du dataset qui est une partie d'un ensemble beaucoup plus gros a été utilisée. Cette partie est composée d'environ 93 000 images de profondeur obtenues grâce à un lidar fixé sur une voiture. Seules les images "non traitées" sont disponibles et présentent énormément de trous. Toutes les images sont des images d'extérieur.

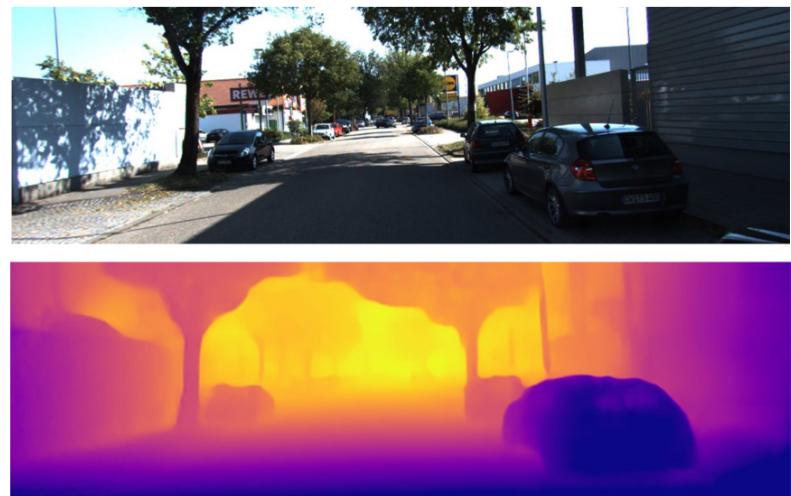


FIGURE 2.5 – Image du *dataset KITTI* (en haut) avec la carte de profondeur correspondante (en bas)

Ces *datasets* ont été choisis pour leur grande taille. Ils ont cependant le désavantage d'être spécialisés dans un seul type d'environnement, l'intérieur pour NYU V2 et l'extérieur depuis une voiture pour KITTI. Il a fallu choisir de travailler sur un seul de ces deux *datasets* pour des raisons d'espace de stockage. Les images d'intérieur ont été choisies car le dataset NYU V2 était plus complet et plus diversifié que le dataset KITTI. Une généralisation aux deux types d'images sera présentée dans la section 3.2.3.

Chapitre 3

Contribution

3.1 Structure globale

Cette section présente la structure globale de l'algorithme final. Chacun des scripts de l'algorithme sera expliqué en détail dans les chapitres suivants. La figure 3.1 présente un schéma détaillé de l'architecture du code. Il est composé de huit fichiers principaux et se déroule en trois grandes parties : une partie de *Machine Learning* pour récupérer les prédictions des différents modèles qui sera développée dans les chapitres 3.2 et 3.3, une partie de traitement d'image pour améliorer les sorties précédentes qui sera développée dans le chapitre 3.3 et enfin une partie d'optimisation permettant à la partie estimation de profondeur de trouver les plans optimaux dans l'image en s'aidant des objets identifiés dans celle-ci qui sera développée dans le chapitre 3.4.

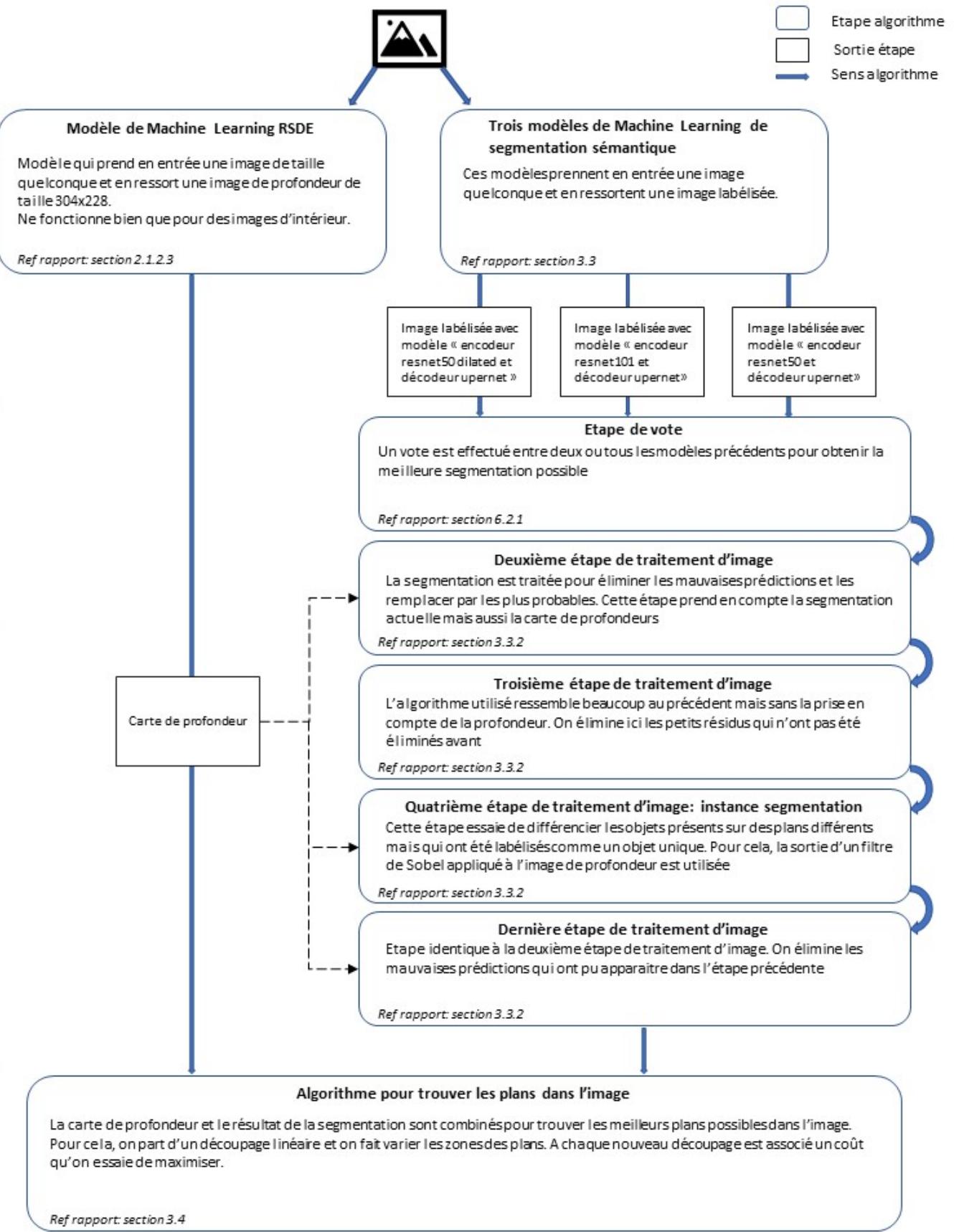


FIGURE 3.1 – Structure globale du projet

3.2 Estimation de profondeur

Ce chapitre présentera une analyse approfondie et précise du modèle RSDE. Cette analyse avait pour objectif de comprendre les forces et les faiblesses de celui-ci, de pouvoir le comparer avec d'autres modèles et de l'améliorer.

3.2.1 Carte des erreurs

Les résultats quantitatifs du modèle ne sont pas bons. Pourtant, visuellement, les résultats sont très corrects. En effet, comme on le voit sur la figure 3.2, on reconnaît très bien les contours des objets et les éléments de la scène en général.



FIGURE 3.2 – Entrée et sortie du modèle RSDE

De plus, les valeurs de profondeur ressorties sont cohérentes. Cette contradiction entre le quantitatif et le qualitatif peut être expliquée par le nombre de valeurs à prédire. Dans beaucoup de modèles, une seule classe de sortie doit être prédite. Ici, plus de 300 000 pixels doivent être correctement estimés par image.

Cela a donné lieu à une première analyse de la façon dont le modèle se comporte. Celle-ci avait pour but d'essayer de comprendre s'il existe un *pattern* à la précision du modèle. Par exemple, peut-être que le fond est toujours mal prédit et fausse les résultats quantitatifs. Pour cela, des cartes des erreurs de prédictions de profondeur ont été créées en prenant la valeur absolue de la différence entre la profondeur des pixels de l'image de référence et la profondeur des pixels de l'image prédite. Ainsi, plus les pixels seront clairs sur l'image créée et plus la différence sera grande et plus les pixels seront foncés sur l'image créée et plus la différence sera faible. Un exemple est présenté dans la figure 3.3. L'analyse a été effectuée sur de très nombreuses images différentes, en y intégrant toutes les situations compliquées comme la présence de vitres ou de miroirs.

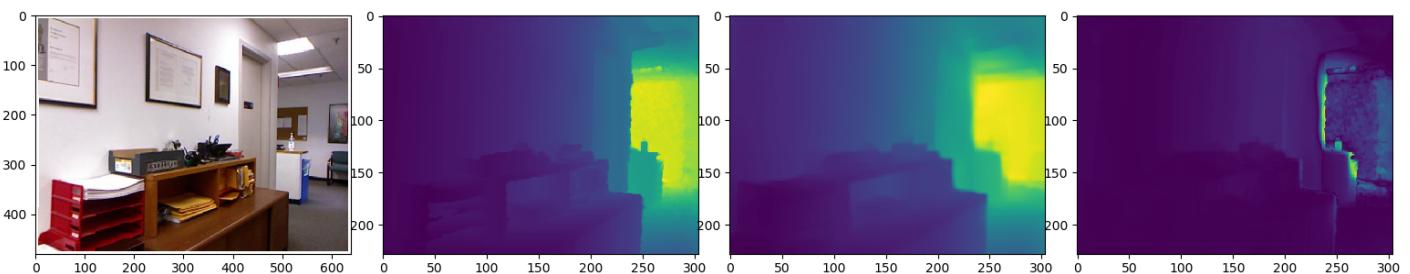


FIGURE 3.3 – Carte des erreurs sur une image. De gauche à droite : image originale, carte de profondeur de référence, carte de profondeur prédite, carte des erreurs.

Cette étude a permis de mettre en avant plusieurs résultats. D'abord sur les endroits des erreurs. En effet, ce sont toujours les plans les plus éloignés qui sont les moins bien prédits. Ensuite sur les types d'images les mieux prédits. Il a été mis en exergue que les images de type "couloir de profondeur" étaient toujours très bien prédites. Finalement, des problèmes liés à l'utilisation de lasers pour l'obtention des cartes de profondeur ont été pointés du doigt. En effet, les vitres ainsi que les pièces de plus de huit mètres de profondeur (distance maximale captée par la kinect utilisée) posent des problèmes au modèle.

Cette partie a été très utile pour la suite du projet. Elle a permis d'orienter les améliorations et de prendre en compte les faiblesses du modèle.

Dans la suite du chapitre, les résultats seront tous obtenus à partir de l'image de gauche de la figure 3.3. Cela permettra au lecteur de comparer les résultats entre eux et de pouvoir observer la progression de l'algorithme.

3.2.2 Découpage de l'image en plans

Découpage linéaire

Plusieurs manières de découper l'image en plans ont été essayées. La première est un découpage linéaire. Il s'agit d'un découpage qui consiste à diviser linéairement la carte de profondeur en n parties égales. Cette méthode a l'avantage d'être simple mais ne permet pas de choisir automatiquement un nombre de plans optimal. Cependant, cette méthode est celle qui se rapproche le plus de la manière dont un humain découperait une image en plans si l'image est peu profonde. En effet, on aurait tendance à vouloir partager équitablement la profondeur entre tous les plans. La figure 3.4 présente des résultats du découpage linéaire.

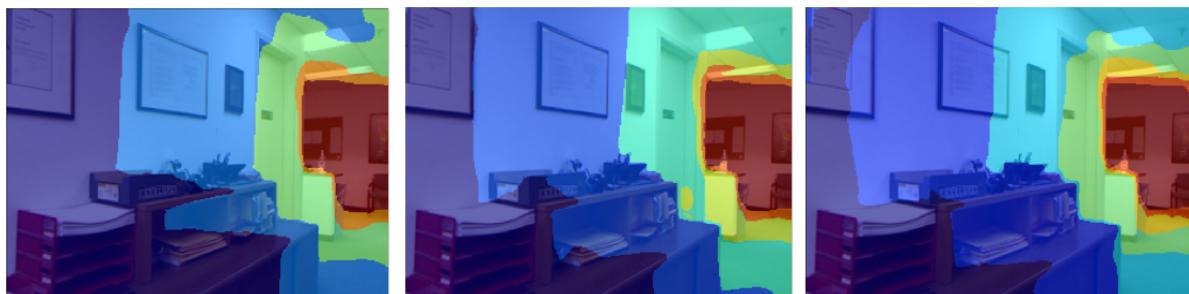


FIGURE 3.4 – Découpage linéaire de l'image en 5, 6 et 7 plans

Découpage log-linéaire

Ce découpage consiste à passer la carte de profondeur dans le domaine logarithmique puis de la découper linéairement. Cette méthode est une fois de plus très simple mais ne permet pas de choisir le nombre optimal de plans dans l'image. Elle est cependant intéressante puisqu'elle se rapproche de la manière dont un humain découperait une image en plans si l'image est assez profonde. En effet, on aurait tendance à privilégier les premiers plans au détriment de ceux plus profonds. La figure 3.5 présente des résultats du découpage log-linéaire.



FIGURE 3.5 – Découpage log-linéaire de l'image en 5, 6 et 7 plans

Découpage par clustering

Ce découpage se base sur l'algorithme K-Means [2] pour découper l'image. C'est une méthode non supervisée qui permet de trouver k *clusters*. L'algorithme de cette méthode est le suivant :

Algorithm 1 Méthode des centres mobiles

- 1: Initialisation de l'algorithme en sélectionnant k individus, appelés centres initiaux, appartenant aux clusters $C_1^0, C_2^0, \dots, C_k^0$.
 - 2: Selection du type de distance entre les individus qui sera utilisé par la méthode (distance euclidienne par exemple).
 - 3: **Constitution des classes** : On répartit l'ensemble des individus en k clusters $\Gamma_1^0, \Gamma_2^0, \dots, \Gamma_k^0$. Un individu est assigné à un cluster Γ_i^0 pour $i = 1, \dots, k$ s'il est plus proche en terme de distance du centre C_i^0 que n'importe quel autre centre C_j^0 pour $j = 1, \dots, k$ et $j \neq i$
 - 4: **Calcul des nouveaux centres** : On détermine le centre de gravité pour chacun des k clusters et on le définit comme nouveau centre de la classe. En d'autres termes pour chaque cluster Γ_i^0 avec $i = 1, \dots, k$ on assigne son centre de gravité à C_i^1 .
 - 5: **On répète l'étape 3 et 4 jusqu'à convergence**
-

Il s'agit d'un algorithme rapide et efficace. Cependant, il nécessite de connaître le nombre de *clusters* à extraire. Un autre algorithme a été utilisé de pair avec l'algorithme K-Means. Il s'agit de l'algorithme des silhouettes [21]. Il permet de déterminer le nombre optimal de *clusters* et donc le nombre optimal de plans dans une carte de profondeur. Cet algorithme associe un score appelé "score des silhouettes" à un découpage donné. Ce score varie entre -1 et 1 et représente la cohésion de points de chaque *cluster* par rapport à leur distance aux autres *cluster*. Plus le

score sera élevé et plus le découpage, donc le nombre de plans choisis pour l'image, sera bon. On calcule ce score pour plusieurs nombres de plans et on choisit celui qui donne le meilleur score des silhouettes. L'algorithme de cette méthode est le suivant :

Algorithm 2 Méthode des silhouettes

- 1: Sélection du type de distance d entre les individus qui sera utilisé par la méthode (distance euclidienne par exemple).
- 2: Pour chaque point i de chaque $cluster C_i$, on calcule sa distance moyenne aux autres points du même $cluster$:

$$a(i) = \frac{1}{|C_i|-1} \sum_{j \in C_i, i \neq j} d(i, j)$$

Plus $a(i)$ sera petit et plus i sera assigné au bon cluster.

- 3: On calcule ensuite sa distance moyenne à tous les points d'un autre $cluster$. On répète cette opération pour chaque autre $cluster$ et on choisit la valeur minimum :

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

Plus $b(i)$ sera grand et plus i serait mal classé s'il avait été classé dans un autre cluster.

- 4: On calcule la valeur des silhouettes pour chaque point i :

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & \text{si } a(i) < b(i) \\ 0, & \text{si } a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, & \text{si } a(i) > b(i) \end{cases}$$

- 5: Une fois $s(i)$ calculé pour chaque point i , on calcule le score des silhouettes du découpage considéré :

$$SC = \max_k \tilde{s}(k)$$

avec $\tilde{s}(k)$ la moyenne des valeurs de silhouette des points pour une nombre spécifique de $cluster$ k .

Cet algorithme permet donc de choisir automatiquement le nombre optimal de plans dans l'image. L'inconvénient de celui-ci est qu'il est beaucoup plus lent que les algorithmes précédents. La figure 3.6 présente le résultat du découpage par clustering.

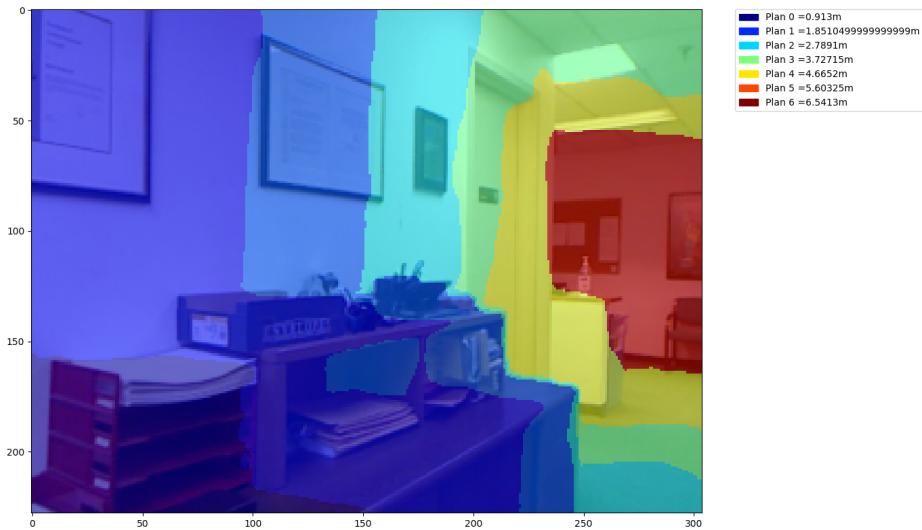


FIGURE 3.6 – Découpage de l'image grâce au clustering

Ces trois méthodes donnent des résultats très proches. Durant le projet, le choix a été fait de détecter toujours trois plans dans une image. Ce choix a permis de résoudre le problème des deux premiers algorithmes qui n'étaient pas capables de déterminer automatiquement un nombre de plans dans l'image. Le dataset de travail choisi étant composé d'images d'intérieur, donc d'images peu profondes, c'est le découpage linéaire qui a été choisi.

3.2.3 Généralisation du modèle

En *Machine Learning*, une caractéristique importante des modèles à prendre en compte est leur généralisation sur des environnements inconnus. Le modèle, ayant été entrainé sur le *dataset* NYU V2 et donc des images d'intérieur, a été testé sur des images d'extérieur. Les résultats furent mauvais. L'idée était donc de réussir à généraliser le modèle à d'autres types d'images en l'entraînant sur un *dataset* comprenant plus d'environnements différents. Pour ce faire, un nouveau *dataset* composé d'images du *dataset* NYU V2 et du *dataset* KITTI a été créé. En raison de la période de télétravail, le *dataset* devait avoir une taille maximum de 14 go pour être entraînable sur Google Colaboratory. Le choix a été fait de se limiter à un entraînement sur environ 30 000 images du *dataset* NYU V2 et 10 000 images du *dataset* KITTI. Cette répartition a été choisie en essayant de maximiser le nombre d'images tout en gardant une majorité d'images du *dataset* NYU V2 mais sans mettre trop peu d'images KITTI. En effet, il fallait essayer de détériorer le moins possible les résultats actuels. Cela permet d'arriver à un volume de 12.1 go de données pour l'entraînement.

Les résultats furent mitigés. En effet, les images du *dataset* NYU V2 sont un peu moins bien prédites après cet entraînement. C'est un résultat logique puisque le modèle s'est entraîné sur 20 000 images du *dataset* NYU V2 en moins. Les images du *dataset* KITTI ont été un peu mieux prédites mais sont restées inexploitables. Cet entraînement a cependant permis de démontrer qu'il était potentiellement possible de généraliser le modèle aux images des deux *datasets*. Pour vérifier cela, un *dataset* de plus grande envergure sera nécessaire comme par exemple 50 000 images NYU V2 et 50 000 images KITTI. Cette étude a ouvert de nouveaux axes de réflexions et de recherches sur le sujet.

3.2.4 Nouvelle architecture

Dans cette partie, les différents travaux effectués pour améliorer le réseau de neurones DenseDepth seront expliqués. C'est celui-ci qui a été sélectionné pour être amélioré plutôt que le réseau RSDE pour deux raisons. D'abord parce que, comme on peut l'observer sur la figure 2.1, c'est celui qui présente les meilleurs résultats. Essayer d'améliorer un autre l'aurait peut-être juste amené à égaler le DenseDepth, ce qui n'aurait pas apporté de plus-value à l'algorithme final. La deuxième raison est son nombre de paramètres. Avec ses 42,6 millions de paramètres, il nécessite l'entraînement d'environ 50% de paramètres en moins que le RSDE. L'architecture DenseDepth a donc potentiellement une plus grande marge de manœuvre avant d'atteindre l'*overfitting* et il est donc plus pertinent de travailler avec. Ce travail a été réalisé à la fin du stage et une seule nouvelle architecture a été essayée. L'idée était de rajouter des couches de *batch-normalization* [13] et de *dropout* dans la partie décodeur qui n'en possède pas. Ces couches sont en effet recommandées dans les réseaux de neurones du fait de leurs très bons résultats. Dans le décodeur, une couche de *batch-normalization* a donc été ajoutée après chaque couche de convolution et une couche de *dropout* avec une probabilité p de 50% a été ajoutée après chaque couche d'activation ReLU. Ces deux couches vont être explicitées ci-après.

Batch-Normalization

En *Machine Learning*, on normalise très souvent les données avant de les entrer dans le modèle. Cela permet d'empêcher que certaines données ayant des échelles plus grandes que les autres n'aient un poids plus important que les autres. Cela permet aussi d'augmenter les performances en augmentant la vitesse de convergence du modèle. C'est en observant la puissance de ce principe que la *batch-normalization* a été créée. L'idée est de se rendre compte que malgré la normalisation en entrée, certains neurones à l'intérieur du réseau peuvent donner des résultats bien plus grands que les autres et on peut ainsi retomber dans le travers évité par la normalisation. On va alors éviter cela en normalisant les résultats intermédiaires à l'intérieur du réseau. Comme avec la normalisation, cela permet d'améliorer les performances du modèle.

En pratique, on effectue cette normalisation sur chaque *mini-batch* c'est-à-dire sur chaque petit lot de données entré dans le modèle. Pour un *mini-batch* $B = \{x_1, x_2, \dots, x_m\}$ donné on commence par calculer :

- Sa moyenne : $\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$
- Sa variance : $\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$

Ensuite, pour chaque neurone du réseau où on applique la *batch-normalization*, on normalise ses entrées $(x^{(1)}, x^{(2)}, \dots, x^{(d)})$:

$$\hat{x}_i^{(k)} = \frac{x_k^{(i)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)2} + \epsilon}}$$

avec $k \in [1, d], i \in [1, m]$

On pourrait s'arrêter là mais alors chaque $\hat{x}_i^{(k)}$ aurait une moyenne nulle et une variance de 1. Ce n'est pas forcément ce que l'on veut. On effectue alors une dernière étape en calculant :

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}.$$

Les paramètres $\gamma^{(k)}$ et $\beta^{(k)}$ seront appris par le modèle durant l'entraînement. C'est sur $y_i^{(k)}$ qu'est appliquée la fonction d'activation du neurone.

Dropout

Le principal problème du *Machine Learning* est l'*overfitting* qui peut très vite arriver. Pour régler ce problème, beaucoup de techniques ont été développées. L'une d'entre elles est le *dropout* [24]. Cette technique consiste, à chaque itération lors de l'entraînement, à désactiver chaque neurone avec une probabilité p . Ainsi, à chaque itération, le modèle sera un peu différent. Cela permet de créer une instabilité qui oblige le réseau à devoir toujours s'adapter, empêchant certains neurones à seulement apprendre à corriger les autres. Les résultats de cette technique sont probants et elle fait maintenant partie des briques de base d'un réseau de neurones.

Résultats

L'entraînement réalisé n'a pas permis d'améliorer les résultats. Le modèle s'est même dégradé en obtenant un score RMS de 0.61. Par manque de temps, la recherche d'une amélioration de l'architecture du modèle n'est pas allée plus loin. Cependant, plusieurs raisons peuvent être la cause de ce résultat. La nouvelle architecture peut en être la raison. Les hyperparamètres optimaux ont aussi pu changer et il aurait fallu faire une recherche de ceux-ci. L'optimiseur utilisé peut aussi devoir être changé pour s'adapter aux nouvelles couches.

3.3 Segmentation d'image

3.3.1 Choix

L'objectif de cette partie est double. D'abord, détecter précisément les objets dans une image permettrait de pouvoir la décrire encore plus précisemment. De plus, comme expliqué dans le chapitre suivant, cette détection permettra d'améliorer la détection des plans dans l'image.

Plusieurs types d'algorithmes de détection d'objets existent :

Les algorithmes de détection d'objet : ces algorithmes entourent les objets détectés dans des boîtes et prédisent leur type. Le modèle de ce type le plus utilisé est le modèle YOLO [20]. Ces algorithmes ont l'avantage de donner de très bons résultats mais il sont peu précis. En effet, une partie de chaque boîte dessinée est en dehors de l'objet en question, comme le montre la figure 3.7

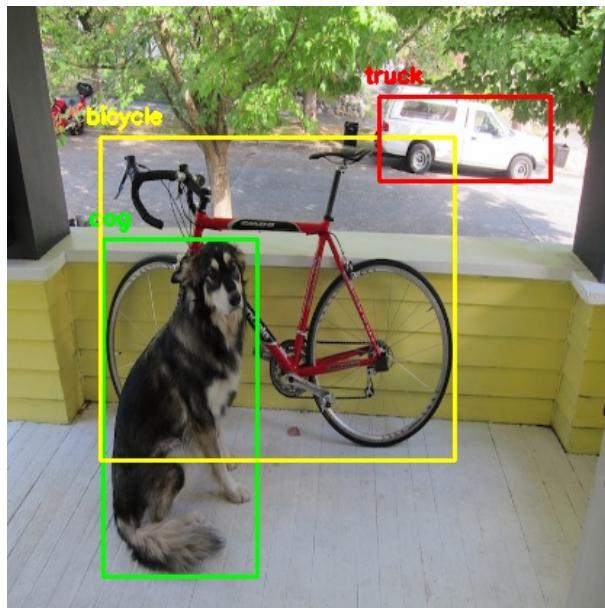


FIGURE 3.7 – Exemple de sortie d'un algorithme de détection d'objet. On observe qu'avec cette méthode, beaucoup de pixels de l'image sont mal prédits. Par exemple, une partie du fond de l'image est classifiée comme bicyclette. Image provenant de [20]

Les algorithmes de segmentation sémantique : Ce type d'algorithme permet de détecter les contours précis des éléments d'une image. Elle divise l'image en différentes classes mais ne permet pas de différencier différentes instances à l'intérieur d'une même classe. Si l'image comporte dix personnes, ces dix personnes seront considérées comme une même entité, comme le montre la figure 3.8. Ce type d'algorithme donne à l'heure actuelle de moins bons résultats que la détection d'objets en raison de sa plus grande complexité.

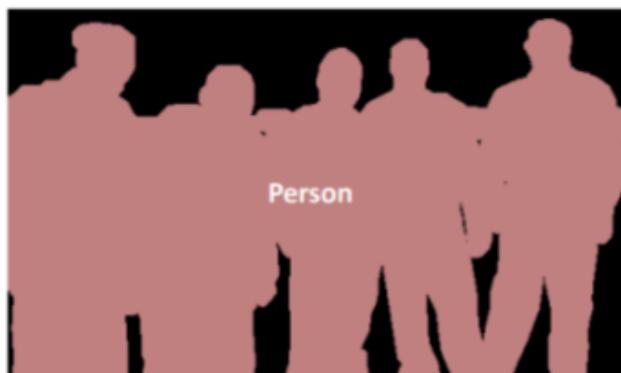


FIGURE 3.8 – Exemple de sortie d'un d'algorithme de segmentation sémantique. Image tirée de : <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>

Les algorithmes de segmentation sémantique instanciée : Comme le précédent, ce type d'algorithme permet de détecter les contours précis des éléments dans une image. Cependant, il est capable de différencier les instances à l'intérieur de chaque classe détectée, comme le montre la figure 3.9.

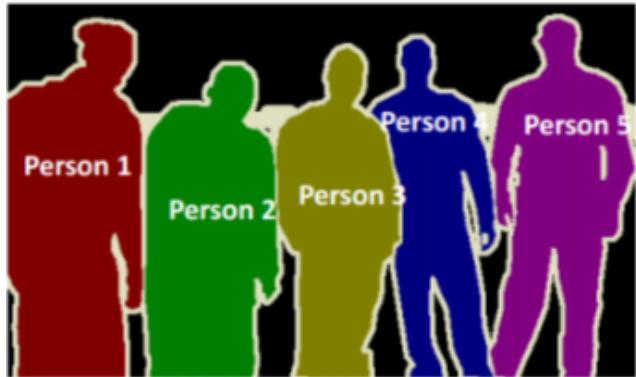


FIGURE 3.9 – Exemple de sortie d'un d'algorithme de segmentation sémantique instanciée. Image tirée de : <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>

C'est la segmentation sémantique qui a été retenue. En effet, la détection d'objets est trop imprécise et ne permettrait pas d'aider la partie détection de plans. La segmentation sémantique instanciée est très complexe et peu de modèles promettent des résultats convenables sur ce type de détection.

Le problème de cette méthode est qu'il était impossible de créer un *dataset* puisque cela impliquerait de labéliser chaque image. Les images étant complexes, une telle entreprise aurait pris plusieurs mois. De plus, contrairement aux images de profondeurs à disposition en grand nombre, uniquement 1449 images labélisées du *dataset* NYU V2 étaient à disposition. Il était donc compliqué d'envisager un entraînement sur si peu de données. La meilleure solution fut de trouver des modèles déjà entraînés. Le *dataset* le plus approprié était alors le *dataset* "Ade20k". En effet, il s'agit d'un *dataset* comprenant environ 20 000 images labélisées, dans lesquelles figurent des images d'intérieur.

Seize modèles ont été essayés. Ils ont à la fois été testés pré-entraînés ainsi que réentraînés par fine-tunage avec le *dataset* NYU V2. Sur l'ensemble de ces modèles, trois donnant des résultats corrects mais toujours inexploitables ont été gardés. Le premier est un modèle de type encodeur-décodeur avec un réseau "resnet50_dilated" en encodeur et un réseau "upernet" [26] en décodeur. Le deuxième est aussi de type encodeur-décodeur avec un réseau "resnet101" en encodeur et un réseau "upernet" en décodeur. Le dernier est toujours de type encodeur-décodeur avec un réseau "resnet50" en encodeur et un réseau "upernet" en décodeur. Les figures 3.10, 3.11 et 3.12 présentent les résultats de ces différents modèles sur l'image utilisée tout au long de ce rapport.



FIGURE 3.10 – Sortie du modèle "encodeur resnet50 dilated et décodeur upernet"



FIGURE 3.11 – Sortie du modèle "encodeur resnet101 et décodeur upernet"

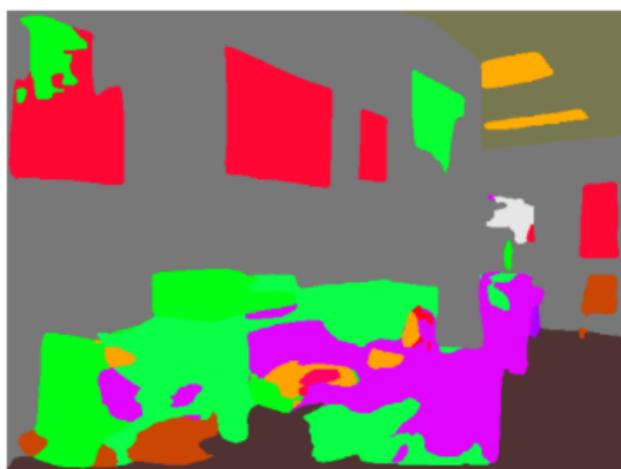


FIGURE 3.12 – Sortie du modèle "encodeur resnet 50 et décodeur upernet"

3.3.2 Amélioration de la segmentation

Comme observé sur les images précédentes, les sorties des modèles de segmentation séquentielle ne sont pas utilisables directement. Cinq étapes de traitement d'images ont été développées pour améliorer leur rendu.

Première étape - vote entre les modèles

Les trois modèles sélectionnés sont plus ou moins bons selon les images et aucun des trois modèles n'est clairement meilleur que les autres. C'est pour cela qu'un vote est mis en place pour essayer de tirer le plus d'informations possibles de chacun d'eux. Deux systèmes de vote ont été mis en place : D'abord, un vote entre deux modèles. Les modèles "encodeur resnet50 dilated et décodeur upernet" et "encodeur resnet101 et décodeur upernet" ont été choisis car ils présentaient des résultats un tout petit peu meilleurs que le troisième. Ce vote est réalisé pixel par pixel entre les deux images obtenues. Il se base sur la confiance que chaque modèle a en sa prédiction tout en essayant de maximiser le nombre d'objets "importants" détectés. En effet, plusieurs classes d'objets ont été jugées "non importants" car trop globaux. Leur détection apporte moins de valeur ajoutée à la détection globale. Ces objets sont les murs, le ciel, le sol, le plafond et les sources de lumière. Ces classes seront toujours défavorisées par rapport à une autre classe dont la confiance est supérieure à 50%. Ainsi, si le premier modèle est confiant à 99% que le pixel x représente un élément "défavorisé" et que le deuxième modèle est confiant à plus de 50% que ce pixel représente une table, le pixel sera quand même labélisé comme une table. Cependant, si le deuxième modèle avait prédit du sol à 70%, c'est le premier modèle qui aurait remporté le vote. On remarque que ce vote est basé sur certains paramètres qui ont été choisis à la fois empiriquement et par intuition. Le deuxième vote est un vote entre les trois modèles sélectionnés. Une fois de plus, les objets sont avantagés au détriment des objets cités plus haut. Les votes tiennent aussi compte de la confiance que chaque modèle a en sa prédiction de telle manière à ce que deux modèles sûrs à 70% en leur prédiction soient avantagés par rapport au troisième si sa confiance n'est pas supérieure à 90%. Ainsi, pour ses caractéristiques de croissance et car $90^{2.75} \approx 2 * 70^{2.75}$, la fonction de vote $f(x_k) = \sum_i x_k^{2.75}$ avec x_k la classe prédictive a été choisie.

L'utilisateur peut choisir d'utiliser l'un des deux votes. Dans la suite de cette section du rapport, les résultats présentés seront ceux obtenus depuis le vote entre les trois modèles. La figure 3.13 présente le résultat de cette première étape de post-traitement.

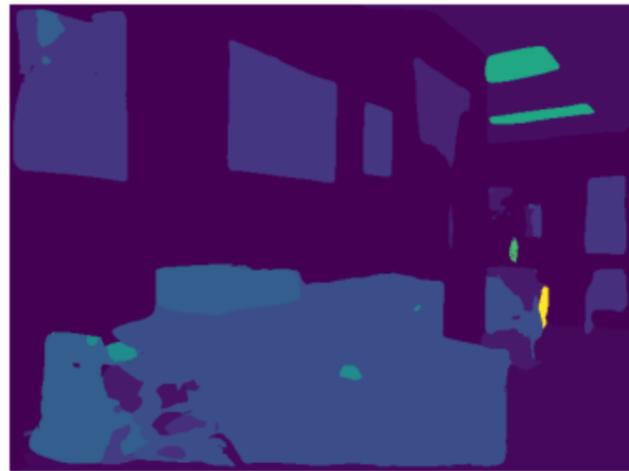


FIGURE 3.13 – Résultat première étape

On observe que l'image est toujours inutilisable et on comprend que les étapes suivantes sont essentielles.

Deuxième étape

Dans cette étape, un algorithme va se charger d'estimer les endroits où les prédictions sont mauvaises et de les corriger. Pour cela, il prend en compte les tailles des prédictions, leur environnement, ainsi que leur distance. En effet, il prend en entrée le résultat de l'étape précédente mais aussi la carte de profondeur retournée par le modèle de détection de profondeur. Cet algorithme sera réutilisé par la suite. C'est aussi ici qu'une modification sur les prédictions du modèle est effectuée. En effet, les modèles de segmentation choisis ont du mal à différencier une chaise d'un fauteuil, ce qui résulte en des prédictions farfelues où une partie d'une chaise est prédite comme étant une chaise tandis qu'un autre partie de celle-ci est prédite comme étant un fauteuil. Pour régler ce problème, les deux prédictions sont rassemblées sous une seule et même prédition : "chaise".

Pour déterminer si une zone a été mal prédite et doit être modifiée, on calcule dans un premier temps la fonction suivante :

$$\text{seuil}(P_g, p) = \frac{1}{\Delta^{\frac{1}{2} * \min(p)}}$$

avec P_g un vecteur contenant les profondeurs de chaque pixel de l'image, p la profondeur de la zone étudiée et $\Delta = \frac{\max(P_g) - \min(P_g)}{2}$.

Cette forme de fonction a été choisie pour ses propriétés d'évolution dépendants de la profondeur de l'image choisie et de la zone de travail actuelle et a été raffinée empiriquement. On compare ensuite le résultat de cette fonction à :

$$t = \frac{\text{taille de la région}}{\text{taille de l'image}} * 100$$

Si t est plus petit que le seuil, alors on juge que la région est mal estimée. On remarque que plus la zone étudiée est profonde et plus le seuil diminue. Cela prend donc en compte le fait que plus les objets sont éloignés et plus ils sont rendus petits dans une image. Un petit objet estimé loin aura donc peu de chance d'être considéré comme mal estimé.

Pour définir la classe par laquelle remplacer la mauvaise prédiction, l'algorithme développé utilise son environnement. Il va d'abord regarder toutes les classes qui l'entourent. Par ordre croissant de taille, il va regarder si leur taille est supérieure à la sienne et si la zone de contact entre les deux zones est supérieure à 30% du périmètre de la zone mal prédite. Ces vérifications ont pour but d'éviter de changer une zone mal prédite en une autre zone mal prédite. Si une classe valide ces critères, elle est choisie comme nouvelle classe. La figure 3.14 présente le résultat de cette deuxième étape de post-traitement.



FIGURE 3.14 – Résultat deuxième étape

On observe que cette étape a largement amélioré la labélisation de l'image.

Troisième étape

Cette étape utilise un algorithme très similaire à l'étape précédente, à l'exception qu'il ne prend pas en compte la profondeur. Il sera utilisé pour éliminer les tous petits résidus qui passent parfois à travers l'algorithme précédent. Sur la figure 3.15, un cercle rouge a été dessiné pour mettre en valeur l'endroit où un changement a eu lieu.



FIGURE 3.15 – Résultat troisième étape

Quatrième étape

Le but est ici de différencier les objets identiques dans des plans différents mais visuellement superposés. Ainsi, l'objectif est de réaliser de la segmentation sémantique instanciée à l'aide de traitement d'image. Actuellement, si deux objets, par exemple des bureaux, sont visuellement superposés mais que l'un est à 1 mètre et l'autre à 5 mètres, l'algorithme ne saura pas faire la différence et labelisera le tout comme un seul et même bureau. Le but étant que cette étape de segmentation serve à améliorer la recherche des plans, un tel comportement est problématique puisqu'il fausse totalement la compréhension de la scène pour l'algorithme final qui pensera alors dans notre exemple qu'il y a un bureau qui a une extrémité à 1 mètre et l'autre à 5 mètres.

Pour essayer de différencier les objets qui sont dans des plans différents, nous allons utiliser un filtre de Sobel [22] sur la carte de profondeurs pour essayer d'en extraire les contours. Ce filtre semble particulièrement adapté puisqu'il se base sur des calculs de gradients et les contours d'objets sont souvent à l'origine de brusques variations de profondeur. Ces contours sont un peu retravaillés pour essayer de ne garder que les contours les plus importants.

Finalement, l'algorithme va croiser la segmentation actuelle de l'image avec la détection de contours pour essayer de comprendre si les objets labélisés sont en fait plusieurs objets différents. Pour ce faire, l'algorithme essaie de trouver des contours générés par le filtre de Sobel à l'intérieur de zones délimitées par la segmentation actuelle. S'il en trouve, l'idée est de tracer la meilleure droite possible entre ces points à l'aide d'une régression linéaire. Ensuite, en utilisant la profondeur de l'image, l'algorithme va essayer de déterminer quels points au dessus de cette droite font partie d'une autre instance de la classe détectée. Cette étape est très dépendante des contours trouvés ainsi que de la qualité de la segmentation actuelle. La figure 3.16 présente le résultat après cette cinquième étape de post-traitement.



FIGURE 3.16 – Résultat quatrième étape

On observe que le bureau a été divisé en deux instances, ce qui était le résultat recherché. Sur la figure 3.17 un trait rouge a été dessiné pour mettre en avant la droite qui a été choisie.

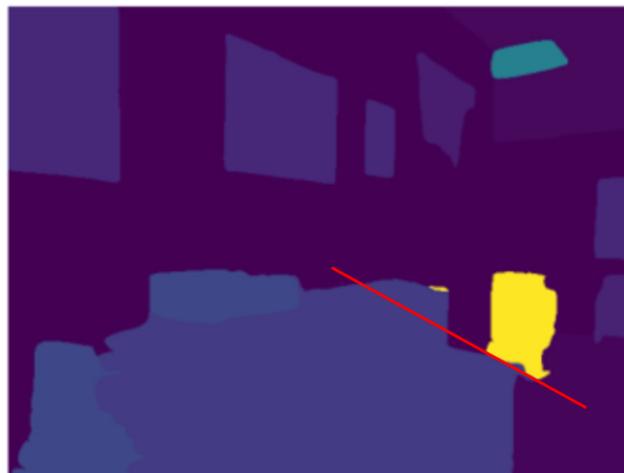


FIGURE 3.17 – Mise en avant de la droite de régression linéaire

On observe que la prise en compte de la profondeur a permis à l'algorithme de comprendre quels pixels devaient être changés d'instance. Sans cela, une grande partie du grand bureau se trouvant au-dessus de la droite aurait changé d'instance, à tort. Ici, seule une petite erreur, proche du bord et donc sûrement due à une mauvaise détection de profondeur, subsiste.

Cinquième étape

Cette étape est identique à la deuxième étape. Elle permet de corriger les possibles erreurs introduites par l'étape précédente. La figure 3.18 montre le résultat de cette étape.



FIGURE 3.18 – Résultat de la cinquième étape

3.4 Optimisation de l'estimation de profondeur

3.4.1 Principe

A ce stade, l'algorithme a généré d'une part une carte de profondeur avec différentes façons de trouver des plans à l'intérieur, et d'autre part une segmentation de l'image. La détection de profondeur a déjà servi à la partie détection d'objet à générer la meilleure segmentation sémantique possible. Le but est ici d'utiliser la segmentation générée pour améliorer la création des plans de profondeur de l'image. La détection des plans de manière linéaire a été sélectionnée pour les raisons évoquées dans la section 3.2.2. Une hypothèse, choisie car cohérente visuellement, est que le découpage linéaire original est plutôt bon et donc qu'on ne va pas énormément le toucher. Une variation de profondeur sur la limite entre deux plans de 10% de la profondeur maximum au plus est autorisée.

Pour réaliser cette optimisation, l'algorithme se base sur le calcul d'une fonction de coût qu'il cherchera à maximiser. Cette fonction de coût augmente pour chaque objet entièrement présent ou presque dans un seul plan. Presque, car un découpage qui coupe 1% ou moins de la taille d'un objet sera rectifié pour obtenir l'objet dans un seul plan. Si deux découpages ont le même coût, c'est celui le plus proche de la configuration initiale qui est choisi. L'algorithme est donc le suivant :

Algorithm 3 Algorithme d'optimisation du découpage en trois plans

- 1: Découpage de l'image en trois plans
 - 2: Calcul de la fonction de coût d'un tel découpage
 - 3: Déplacement d'1% la limite de profondeur entre les plans
 - 4: On répète les étapes précédentes jusqu'à avoir travaillé avec tous les découpages possibles (en gardant la condition de déplacement de 10% maximum)
 - 5: On choisit le découpage ayant le coût maximum
-

3.4.2 Résultats

On observe sur la figure 3.19 qu'il y a finalement peu de différences de découpage en plans avant et après la cinquième étape de l'algorithme. Le découpage s'est tout de même un peu amélioré puisque les éléments du fond de l'image qui étaient sur deux plans ne le sont plus. Ces remarques sont applicables à l'ensemble des images du dataset de test. On peut les expliquer par le fait que le découpage soit réalisé sur trois plans uniquement. Ainsi, le découpage initial est déjà bon et il y a peu de place pour les améliorations. Un découpage en plus de plans aurait sûrement permis d'observer de plus gros changements au passage de cette étape. Un changement de la fonction de coût utilisée permettrait d'obtenir d'autres résultats mais ceux actuels sont convenables.

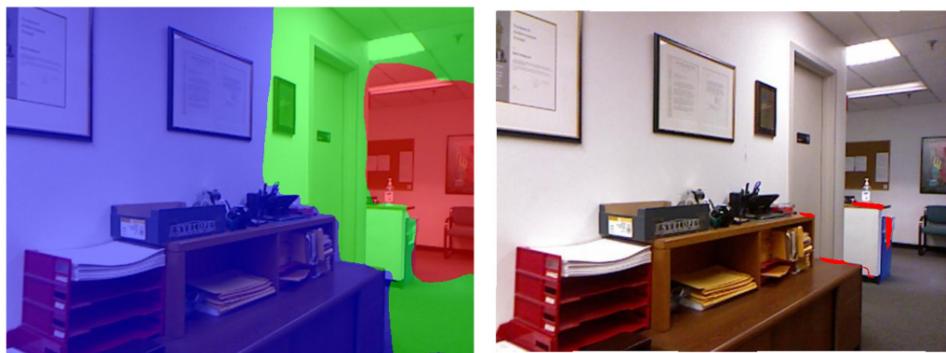


FIGURE 3.19 – Découpage final de l'image en trois plans. A gauche : image sur laquelle les plans ont été dessinés. A droite : la différence de découpage avant et après la cinquième étape de l'algorithme est représentée en rouge.

Chapitre 4

Application

4.1 Présentation

L’application a été réalisée en client lourd, avec la bibliothèque Tkinter de python. Elle possède six fonctionnalités. Le travail réalisé a été le développement de toute l’interface graphique ainsi que les fonctionnalités une et deux. Les cinq autres membres du groupe ayant réalisé les autres fonctionnalités.

4.2 Interface graphique

L’application est composée de deux interfaces principales. La première interface, figurant dans la figure 4.1, permet de sélectionner l’image sur laquelle l’utilisateur veut travailler. Les images proposées sont classées en deux ensembles, les images d’intérieurs et les images d’extérieurs. En effet, certains algorithmes fonctionnent mieux sur des images d’intérieurs et d’autres sur des images d’extérieurs. Cette différenciation permet donc de lancer les meilleurs algorithmes en fonction de l’image choisie. Enfin, si aucune image ne plaît à l’utilisateur, il peut utiliser sa propre image grâce au bouton situé en bas de l’écran.

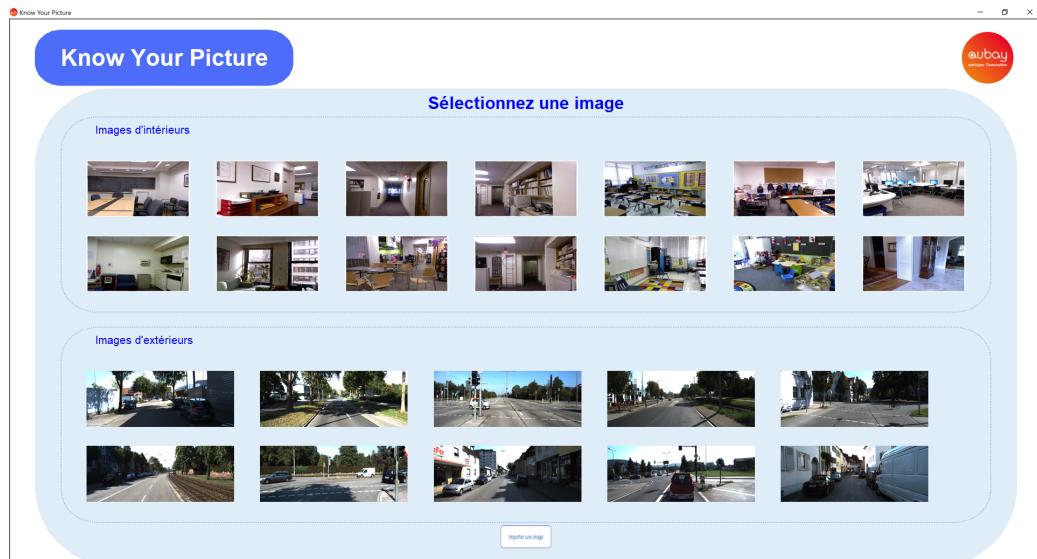


FIGURE 4.1 – Première interface de l’application

Une fois une image sélectionnée, l’utilisateur arrive sur la deuxième interface de l’application, figurant dans la figure 4.2. Elle est composée de quatre éléments principaux. Au centre se trouve la plus grande zone où l’utilisateur pourra observer les résultats des traitements effectués par l’application. Le bandeau vertical à gauche permet de sélectionner la fonctionnalité voulue. Il permet aussi de revenir aux choix des images grâce au bouton situé le plus en haut. Pour rajouter du retour utilisateur et lui permettre de mieux comprendre les différentes fonctionnalités, un court texte apparaît lorsque la souris passe sur un des boutons de fonctionnalité pour la décrire. Les deux derniers bandeaux, en bas et à droite de la fenêtre, permettent d’obtenir des explications ou d’accéder à des options propres à chaque fonctionnalité. Comme on l’observe sur la figure 4.2, au lancement de la fenêtre, l’application indique à l’utilisateur les différents chargements qu’elle est en train d’effectuer.

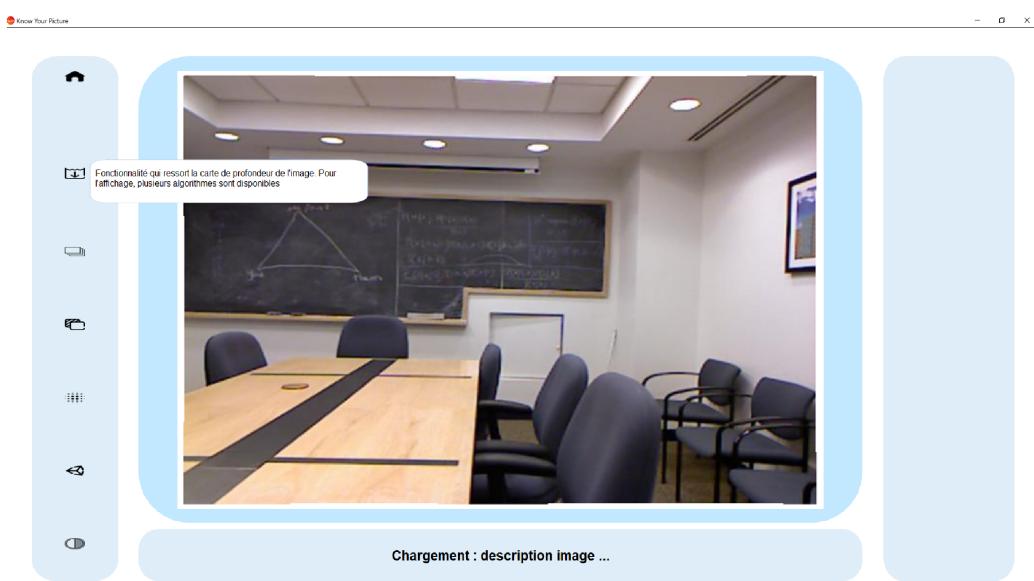


FIGURE 4.2 – Deuxième interface de l’application

4.3 Fonctionnalités

Première fonctionnalité

Cette fonctionnalité permet d'observer les différentes cartes de profondeur ressorties par les différents algorithmes développés. Elle permet à l'utilisateur de savoir très rapidement visuellement si la profondeur de l'image a bien été comprise par les différents algorithmes. Le choix de l'algorithme se fait à l'aide des boutons disponibles dans le bandeau de droite. Une *colormap* a été appliquée à ces cartes de profondeur. Ainsi, plus un pixel est proche, plus il sera violet foncé et plus il sera éloigné, plus il sera orangé. On observe sur la figure 4.3 que l'algorithme a bien compris l'idée de profondeur de cette image. En effet, les objets les plus proches comme les chaises sont bien violet foncé et le tableau en fond d'image est de couleur orangée.



FIGURE 4.3 – Première fonctionnalité

Deuxième fonctionnalité

La fonctionnalité précédente ayant permis de visualiser les résultats de la première partie du projet, la deuxième fonctionnalité permet de visualiser les résultats de la deuxième partie de celui-ci, c'est-à-dire la détection d'objet. Cette fonctionnalité crée une description automatique et précise de l'image, en termes de profondeur mais aussi en termes d'objets présents dans chacun des plans. L'utilisateur a dans un premier temps le choix du plan qu'il veut décrire grâce à des boutons présents à droite, comme on le voit dans la figure 4.4. Une fois le plan choisi, la description commence. L'algorithme présente alors le plan en indiquant la profondeur moyenne de celui-ci. Il continue ensuite en mettant en avant chaque type d'objets en lui superposant du rouge et en indiquant son type. Dans la figure 4.5, on observe que l'algorithme est en train de mettre en avant une table.



FIGURE 4.4 – Deuxième plan choisi

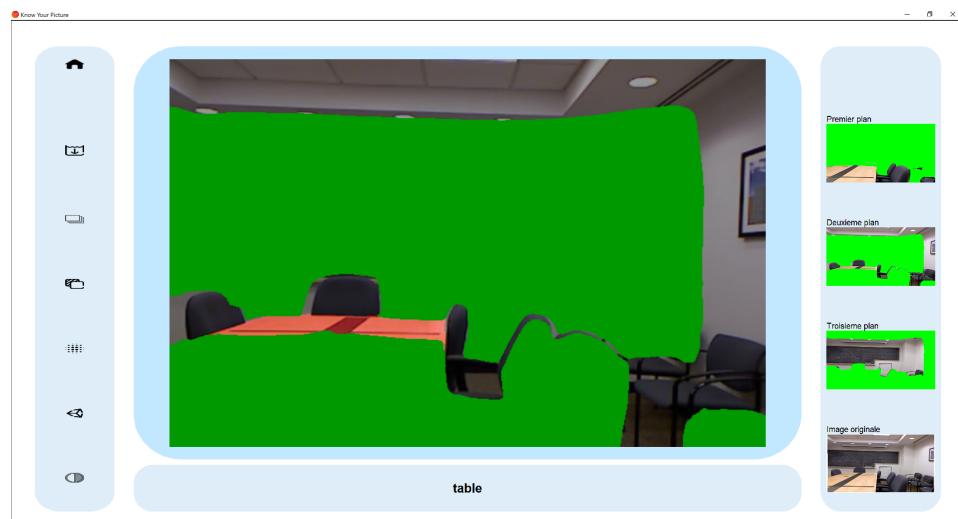


FIGURE 4.5 – Une table a été détectée

Ces deux premières fonctionnalités ont permis de montrer les sorties des algorithmes développés durant ce stage. Les fonctionnalités suivantes permettent de proposer des cas d'utilisation de ces algorithmes.

Troisième fonctionnalité

Cette fonctionnalité permet de remplacer automatiquement le dernier plan de l'image par une autre image. Cela peut être utile dans un contexte de retouches photographiques. Des *sliders* sont présents dans le bandeau droit pour permettre d'ajuster la photo mise au dernier plan. Dans la figure 4.6, on observe que le dernier plan a été remplacé par une image de bibliothèque.

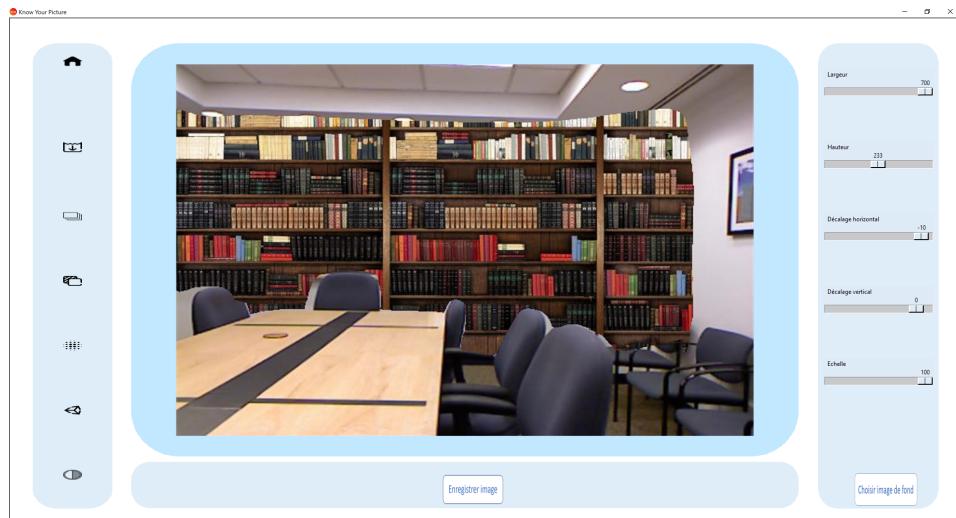


FIGURE 4.6 – Dernier plan remplacé par une bibliothèque

Quatrième fonctionnalité

Cette fonctionnalité permet d'appliquer du flou à l'image. On peut choisir de garder un plan visible et de flouter les autres ou bien de garder visible uniquement un objet. Cette fonctionnalité se rapproche de ce que l'on observe dans l'application Teams pour flouter l'arrière-plan et ne garder que la tête de la personne qui parle. Elle peut donc permettre de mettre en avant quelque chose dans une image. Elle peut aussi être utilisée pour faire de la retouche photographique. Dans la figure 4.7, le premier plan de l'image a été mis en avant et dans la figure 4.8, le tableau a été mis en avant.

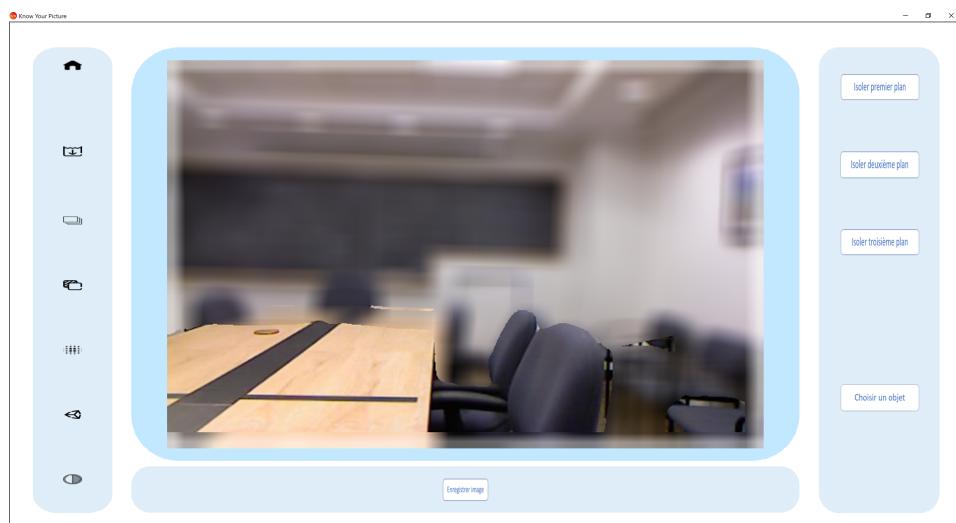


FIGURE 4.7 – Premier plan mis en avant

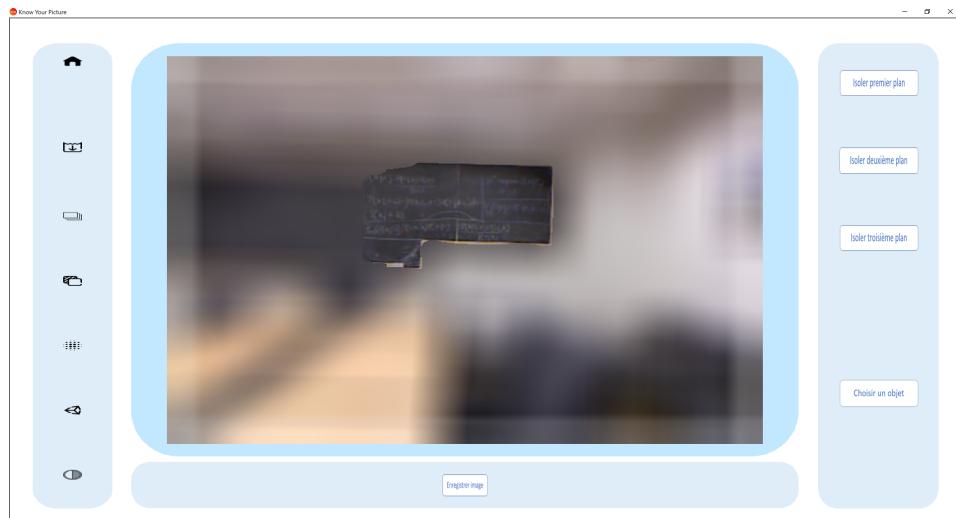


FIGURE 4.8 – Tableau mis en avant

Cinquième fonctionnalité

Cette fonctionnalité permet de mettre en avant les objets en les saturant et en désaturant le reste de l'image. La saturation est aussi appliquée en fonction de la profondeur détectée dans l'image. Ce taux de saturation est réglable grâce à un *slider* présent dans le bandeau droit de l'application. Sur la figure 4.9, on observe que la chaise est saturée et que le reste de l'image est désaturé. De plus, on observe sur la table que plus la profondeur augmente et plus l'image est désaturée. Cette fonctionnalité peut elle aussi permettre de faire des retouches photographiques.

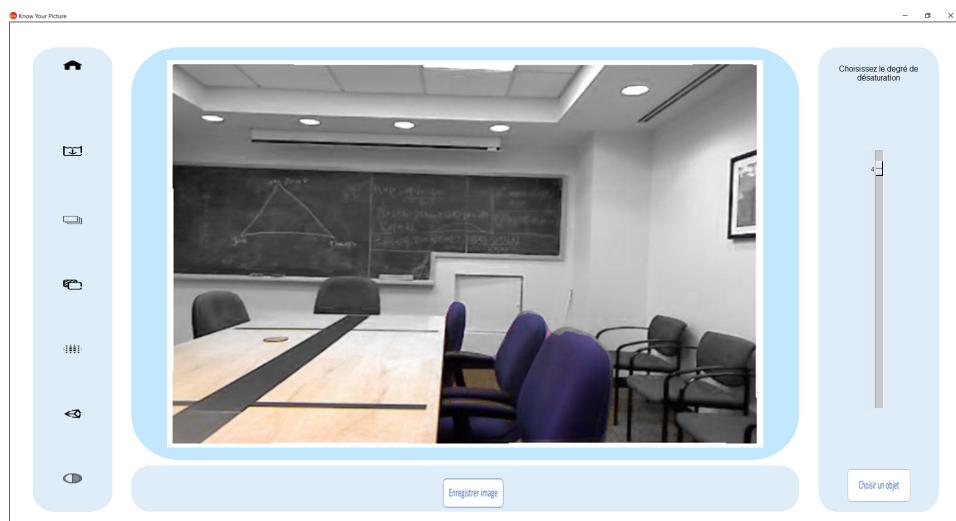


FIGURE 4.9 – Chaises saturées et saturation en fonction de la profondeur

Sixième fonctionnalité

Cette fonctionnalité est la plus visuelle de toutes puisqu'elle permet de créer une image 3D à partir de l'image 2D de départ. Elle génère une fenêtre dans laquelle on peut naviguer à l'aide

de rotations, translations ou zooms. La valeur de profondeur de chaque pixel est utilisée pour générer le nuage de points. Sur la figure 4.10, on observe qu'on a effectué une rotation pour voir l'image d'un autre point de vue. Cela permet de créer une image immersive qui pourrait être utilisée par des architectes, par exemple, pour présenter des photographies d'appartements.

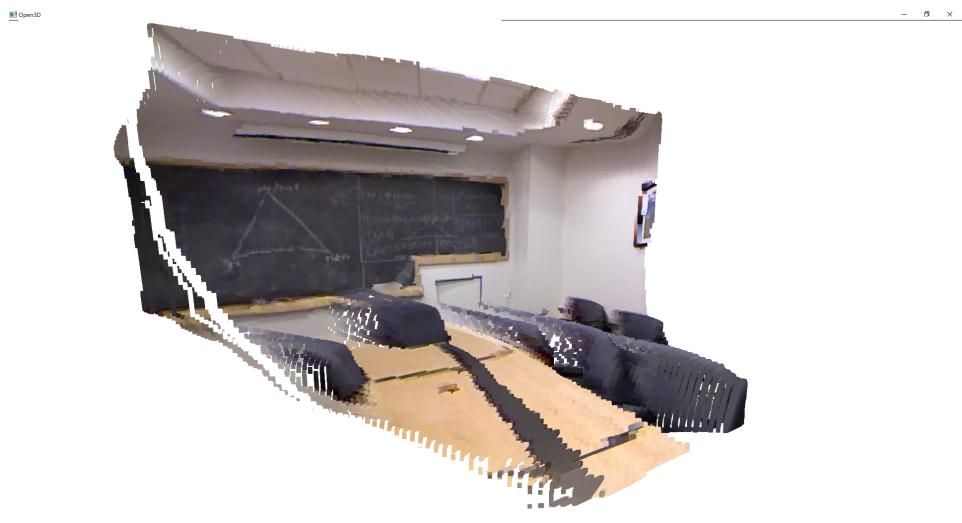


FIGURE 4.10 – Image 3D générée

Conclusion

L'objectif du stage était double. D'une part, il devait permettre de créer un algorithme capable de comprendre le mieux possible une image en termes de profondeur et d'objets présents à l'intérieur. D'autre part il devait permettre de mettre en place des briques de base ainsi que des analyses précises du problème pour l'entreprise AUBAY. Au terme de ces six mois, une application fonctionnelle permettant de décrire une image selon les deux aspects recherchés et présentant des cas d'utilisations pratiques a été développée. Plusieurs algorithmes d'estimation de profondeur ont été mis au point et analysés rigoureusement. Un algorithme de traitement d'image permettant de post-processer efficacement n'importe quelle image segmentée a aussi été créé. L'ensemble de ces travaux permet d'affirmer que l'objectif du stage a été atteint.

Les améliorations principales de ce projet se porteraient sur la suppression des hypothèses de départ. En effet, à l'heure actuelle, les images pouvant être traitées par l'algorithme ne se limitent qu'à deux types d'images. De plus, un algorithme différent gère chacun d'eux. Une première étape serait de créer un algorithme capable de traiter les deux types d'images puis d'élargir les entrées possibles.

Annexe A

Glossaire

Images stéréoscopiques : Ensemble de deux images présentant une même scène de deux points de vue légèrement différents.

Cluster : Petit groupe de points.

Dataset : Jeu de données.

Client lourd : Application qui utilise uniquement les ressources locales de l'ordinateur.

Colormap : Tableau de couleurs permettant d'attribuer une couleur à chaque pixel d'une image.

Pattern : Modèle ou structure répétitive.

Overfitting : Désigne le fait qu'un modèle se soit trop spécialisé sur les données d'entraînement résultant en une mauvaise généralisation.

Post-traitement : Ensemble des opérations réalisées après l'algorithme principal, pour améliorer ses résultats.

YOLO : You Only Look Once est un algorithme de détection d'objets. Il est beaucoup utilisé pour sa simplicité et ses bons résultats.

Skip connections : Connexion entre deux couches d'un réseau de neurones qui ne se suivent pas directement.

Convolution : Opération principale des réseaux de neurones convolutifs et présente dans énormément d'architectures. Elle permet d'extraire simplement des caractéristiques importantes de l'image de travail.

Pooling : Couche d'un réseau de neurone qui permet de diminuer la taille d'une image en concervant les informations importantes. Cela permet de résoudre le problème des couches de convolution qui ont besoin de connaître l'emplacement précis des objets à extraire. Les couches de *pooling* les plus utilisées sont le *max-pooling* qui conserve la valeur maximum des pixels sur lesquels il travaille et l'*average-pooling* qui conserve la moyenne des pixels sur lesquels il travaille.

ReLU : Neurone qui utilise $f(x) = \max(x, 0)$ comme fonction d'activation.

ResNet : Les réseaux de neurones de type *Residual Network* sont des réseaux particuliers qui connectent la sortie d'un neurone à son entrée. Ce type de réseau est utilisé pour résoudre le problème de disparition du gradient qui le fait décroître exponentiellement [8].

SENet : Les réseaux de neurones de type *Squeeze-and-Excitation Networks* sont des réseaux utilisant des blocs de type *squeezing-and-excitation*. Ces blocs permettent d'attribuer un poids à chaque canal de l'image. Ces réseaux présentent d'excellents résultats et n'ajoutent presque aucun coût de calcul [9].

UPerNet : *Unified Perceptual Parsing* est un réseau de neurones spécialisé dans la description d'image. Il est conçu pour déterminer à la fois le type de scène, les objets présents dans celle-ci, les différents éléments les composant et leur matière.

Annexe B

Filtre de Sobel

Le filtre de Sobel est très utilisé en traitement d'image du fait de sa simplicité. Il utilise deux matrices 3x3 qui seront utilisées comme matrices de convolution pour calculer en chaque point une approximation de la dérivée horizontale et verticale. La matrice utilisée pour obtenir une approximation de la dérivée horizontale est celle-ci :

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Pour obtenir une approximation de la dérivée verticale on utilise :

$$M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Les images résultantes sont obtenues grâce aux calculs de G_x et G_y :

$$G_x = M_x * A \text{ et } G_y = M_y * A$$

avec A l'image sur laquelle on applique le filtre Sobel.



FIGURE B.1 – De gauche à droite : image originale, image après filtre de Sobel selon x, image après filtre de Sobel selon y

Finalement, l'image finale est obtenue grâce au calcul de la norme du gradient :

$$G = \sqrt{G_x^2 + G_y^2}$$



FIGURE B.2 – Résultat final du filtre de Sobel

Annexe C

Couches des réseaux DenseNet

| Layers | Output Size | DenseNet-121 | DenseNet-169 | DenseNet-201 | DenseNet-264 |
|----------------------|------------------|--|--|--|--|
| Convolution | 112×112 | | 7×7 conv, stride 2 | | |
| Pooling | 56×56 | | 3×3 max pool, stride 2 | | |
| Dense Block (1) | 56×56 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ |
| Transition Layer (1) | 56×56 | | | 1×1 conv | |
| | 28×28 | | | 2×2 average pool, stride 2 | |
| Dense Block (2) | 28×28 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ |
| Transition Layer (2) | 28×28 | | | 1×1 conv | |
| | 14×14 | | | 2×2 average pool, stride 2 | |
| Dense Block (3) | 14×14 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$ |
| Transition Layer (3) | 14×14 | | | 1×1 conv | |
| | 7×7 | | | 2×2 average pool, stride 2 | |
| Dense Block (4) | 7×7 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ |
| Classification Layer | 1×1 | | | 7×7 global average pool | |
| | | | | 1000D fully-connected, softmax | |

FIGURE C.1 – Architecture des réseaux DenseNet. Chaque couche "conv" correspond à la succession "*batch-normalization*, activation ReLU, convolution"

Table des figures

| | | |
|------|---|----|
| 1.1 | Tableau agile sur le site Jira | 10 |
| 2.1 | Schéma d'une architecture de type encodeur-décodeur | 12 |
| 2.2 | Couches du modèle DenseDepth. Les couches jusqu'à CONV2 sont exactement celles du réseau DenseNet169. Chaque couche CONVB est une couche de convolution suivie d'une activation ReLU [18] | 13 |
| 2.3 | Architecture du modèle RSDE | 14 |
| 2.4 | Image du dataset NYU V2 (à gauche) avec la carte de profondeur correspondante (à droite) | 16 |
| 2.5 | Image du <i>dataset</i> KITTI (en haut) avec la carte de profondeur correspondante (en bas) | 16 |
| 3.1 | Structure globale du projet | 18 |
| 3.2 | Entrée et sortie du modèle RSDE | 19 |
| 3.3 | Carte des erreurs sur une image. De gauche à droite : image originale, carte de profondeur de référence, carte de profondeur prédictive, carte des erreurs. | 20 |
| 3.4 | Découpage linéaire de l'image en 5, 6 et 7 plans | 20 |
| 3.5 | Découpage log-linéaire de l'image en 5, 6 et 7 plans | 21 |
| 3.6 | Découpage de l'image grâce au clustering | 23 |
| 3.7 | Exemple de sortie d'un algorithme de détection d'objet. On observe qu'avec cette méthode, beaucoup de pixels de l'image sont mal prédits. Par exemple, une partie du fond de l'image est classifiée comme bicyclette. Image provenant de [20] | 26 |
| 3.8 | Exemple de sortie d'un d'algorithme de segmentation sémantique. Image tirée de : https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47 | 26 |
| 3.9 | Exemple de sortie d'un d'algorithme de segmentation sémantique instanciée. Image tirée de : https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47 | 27 |
| 3.10 | Sortie du modèle "encodeur resnet50 dilated et décodeur upernet" | 28 |
| 3.11 | Sortie du modèle "encodeur resnet101 et décodeur upernet" | 28 |
| 3.12 | Sortie du modèle "encodeur resnet 50 et décodeur upernet" | 28 |
| 3.13 | Résultat première étape | 30 |
| 3.14 | Résultat deuxième étape | 31 |
| 3.15 | Résultat troisième étape | 32 |
| 3.16 | Résultat quatrième étape | 33 |
| 3.17 | Mise en avant de la droite de régression linéaire | 33 |
| 3.18 | Résultat de la cinquième étape | 34 |
| 3.19 | Découpage final de l'image en trois plans. A gauche : image sur laquelle les plans ont été dessinés. A droite : la différence de découpage avant et après la cinquième étape de l'algorithme est représentée en rouge. | 35 |

| | | |
|------|---|----|
| 4.1 | Première interface de l'application | 37 |
| 4.2 | Deuxième interface de l'application | 37 |
| 4.3 | Première fonctionnalité | 38 |
| 4.4 | Deuxième plan choisi | 39 |
| 4.5 | Une table a été détectée | 39 |
| 4.6 | Dernier plan remplacé par une bibliothèque | 40 |
| 4.7 | Premier plan mis en avant | 40 |
| 4.8 | Tableau mis en avant | 41 |
| 4.9 | Chaises saturées et saturation en fonction de la profondeur | 41 |
| 4.10 | Image 3D générée | 42 |
| B.1 | De gauche à droite : image originale, image après filtre de Sobel selon x, image après filtre de Sobel selon y | 47 |
| B.2 | Résultat final du filtre de Sobel | 47 |
| C.1 | Architecture des réseaux DenseNet. Chaque couche "conv" correspond à la succession " <i>batch-normalization</i> , activation ReLU, convolution" | 48 |

Bibliographie

- [1] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *CoRR*, abs/1812.11941, 2018.
- [2] David Arthur and Sergei Vassilvitskii. K-means++ : The advantages of careful seeding. volume 8, pages 1027–1035, 01 2007.
- [3] Katie Burke. How does a self-driving car see ? <https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/>, 2019.
- [4] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014.
- [5] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. *CoRR*, abs/1806.02446, 2018.
- [6] K.Duan D.Xi Y.Zhu H.Zhu H.Xiong Q.He F.Zhuang, Z.Qi. A comprehensive survey on transfer learning. *arXiv e-prints*, abs/1911.02685, 2020.
- [7] Zhixiang Hao, Yu Li, Shaodi You, and Feng Lu. Detail preserving depth estimation from a single image using attention guided networks. *CoRR*, abs/1809.00646, 2018.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [9] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.
- [10] Junjie Hu, Mete Ozay, Yan Zhang, and Takayuki Okatani. Revisiting single image depth estimation : Toward higher resolution maps with accurate object boundaries. *CoRR*, abs/1803.08673, 2018.
- [11] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [12] Peter Wonka Ibraheem Alhashim. High quality monocular depth estimation via transfer learning. <https://github.com/ialhashim/DenseDepth>, 2018.
- [13] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [14] Yan Zhang Takayuki Okatani Junjie Hu, Mete Ozay. Revisiting single image depth estimation : Toward higher resolution maps with accurate object boundaries. https://github.com/JunjH/Revisiting_Single_Depth_Estimation, 2019.
- [15] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. *CoRR*, abs/1606.00373, 2016.

- [16] Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks : Mixed, gated, and tree, 2015.
- [17] Todd Alexander Litman. Autonomous vehicle implementation predictions : Implications for transport planning. *Victoria Transport Policy Institute*, 2020.
- [18] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.
- [19] Digital Watch Observatory. The rise of autonomous vehicles. <https://dig.watch/trends/rise-autonomous-vehicles>, 2019.
- [20] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [21] Peter Rousseeuw. Rousseeuw, p.j. : Silhouettes : A graphical aid to the interpretation and validation of cluster analysis. *comput. appl. math.* 20, 53-65. *Journal of Computational and Applied Mathematics*, 20 :53–65, 11 1987.
- [22] Darma Setiawan Putra, Mina Muhamarratul, Risky, and Asmaidi Asmaidi. Implementation of sobel method based edge detection for flower image segmentation. *SinkrOn*, 3, 04 2019.
- [23] Wen-Liang Hwang Shih-Shuo Tung. Depth extraction from a single image and its application. <http://dx.doi.org/10.5772/intechopen.84247>, 2019.
- [24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56) :1929–1958, 2014.
- [25] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter : On the transferability of adversarial examples generated with resnets, 2020.
- [26] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. *CoRR*, abs/1807.10221, 2018.
- [27] Dan Xu, Elisa Ricci, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Multi-scale continuous crfs as sequential deep networks for monocular depth estimation. *CoRR*, abs/1704.02157, 2017.
- [28] Jong Chul Ye and Woon Kyoung Sung. Understanding geometry of encoder-decoder cnns. *CoRR*, abs/1901.07647, 2019.