Détection d'objets industriels à l'aide de modèles 3D dans des images égocentriques

Julia Cohen



Université Lumière Lyon 2
Ecole Doctorale : ED 512 Informatique et Mathématiques
LIRIS

THÈSE
pour obtenir le grade de
DOCTEUR EN INFORMATIQUE

Détection d'objets industriels à l'aide de modèles 3D dans des images égocentriques

Julia COHEN

Informatique

Sous la direction de Laure TOUGNE RODET
Co-encadrée par Carlos CRISPIM-JUNIOR

Soutenue publiquement le 13 Juillet 2022

Composition du jury :
Pascal DESBARATS, Professeur des Universités, Université de Bordeaux, *Rapporteur*
Julien MILLE, Maître de conférences (HDR), INSA Centre - Val de Loire, *Rapporteur*
Elisa FROMONT, Professeur des Universités, Université de Rennes, *Examinatrice*
Mathieu AUBRY, Maître de conférences (HDR), Ecole des Ponts ParisTech, *Examinateur*
Laure TOUGNE RODET, Professeur des Universités, Université Lyon 2, *Directrice de thèse*
Carlos CRISPIM-JUNIOR, Maître de conférences, Université Lyon 2, *Co-encadrant de thèse*
Jean-Marc CHIAPPA, Ingénieur, Groupe DEMS St Bonnet de Mûre, *Tuteur entreprise*

Remerciements

Je tiens tout d'abord à remercier mes encadrants de thèse, qui m'ont accompagnée tout au long des dernières années : Laure Tougne et Carlos Crispim-Junior, pour leur expertise scientifique et leur suivi presque quotidien, ainsi que leurs retours fournis sur chacun de mes manuscrits. Je remercie également Céline Grange-Faivre pour avoir mis en place et soutenu ce projet de thèse au sein de DEMS, ainsi que Jean-Marc Chiappa pour avoir pris la suite.

Je remercie mes collègues, au laboratoire comme à l'entreprise, pour les discussions partagés lors des pauses, qu'elles aient mené à de nouvelles idées ou à des éclats de rire bienvenus. Un grand merci également à mes proches pour leur soutien sans faille, pour m'avoir permis de déconnecter autant que nécessaire.

Ces remerciements ne seraient pas complets sans mentionner tous les chercheurs que j'ai pu croiser, écouter ou rencontrer, lors de séminaires ou de conférences, en virtuel ou en réel. De par leur passion à faire et à partager leur recherche, ils ont amplifié ma curiosité, fait germer des idées, et suscité mon admiration.

Enfin, je remercie l'ANRT, DEMS et le LIRIS, pour avoir financé les travaux menés dans cette thèse CIFRE.

Résumé

L'assemblage de produits industriels peut aujourd'hui être facilité et accéléré par l'usage de solutions numériques innovantes telles que la réalité augmentée (RA). En effet, le développement de nouveaux supports tels que des casques de RA permet aux opérateurs de visualiser des instructions tout en ayant les mains libres pour la manipulation des pièces. La détection de ces objets industriels par une caméra positionnée sur le casque permet une adaptation des éléments virtuels à la scène réelle. Cependant, les images issues d'un casque de RA présentent des difficultés inhérentes à leur point de vue égocentrique. Bien que la détection d'objets dans des images soit l'une des applications dans lesquelles l'apprentissage profond excelle, les réseaux de neurones artificiels sont rarement appliqués aux images égocentriques et contenant des objets industriels. En particulier, la tâche se complique lorsqu'aucune image réelle des objets à identifier n'est disponible, et lorsque l'algorithme de détection doit être déployé sur un système embarqué pour une application en temps réel. Dans cette thèse menée au laboratoire LIRIS et en partenariat avec le bureau d'études en ingénierie et design DEMS, nous nous sommes attaqués à la problématique de la reconnaissance d'objets industriels à partir des images d'un casque de RA. Nous avons tiré parti de la disponibilité des modèles 3D des objets d'intérêt afin de générer un jeu de données synthétique égocentrique pour l'entraînement de réseaux de neurones compacts, dédiés à la détection mobile et en temps réel. Nous avons analysé les éléments de ce jeu de données permettant de se passer totalement d'images réelles pour entraîner ce réseau de neurones. Par la suite, nous avons étudié la possibilité d'utiliser l'information de profondeur contenue dans les images RGB-D afin d'améliorer la performance du détecteur d'objets. Nous avons ainsi abordé la problématique de la généralisation de domaine entre des images RGB-D synthétiques et réelles, et nous avons proposé différentes approches afin de réduire l'écart à la réalité, compatibles avec une inférence mobile et en temps réel.

Mots-clés : détection d'objets, apprentissage profond, jeu de données synthétique, point de vue égocentrique.

Abstract

Industrial manufacturing can be facilitated using innovative digital solutions such as Augmented Reality (AR). The development of new devices such as AR headsets and head-mounted devices enable operators to visualize assembly instructions while having their hands free to manipulate the physical pieces. The detection of these industrial objects through a head-mounted camera enables the virtual elements to automatically adapt to the real scene. However, images captured with an AR headset present visual artefacts inherent to the egocentric point of view. Although object detection in images is a popular application of deep learning for its effectiveness, artificial neural networks are rarely applied to egocentric images and industrial objects. The task is even more complex when no real image of the objects of interest is available, and the algorithm will be embedded in a mobile computer with a real-time inference requirement. In this PhD prepared at LIRIS lab and in collaboration with engineering and design company DEMS, we addressed the topic of industrial objects recognition in images from an AR headset. We leveraged the available 3D models of the objects of interest in order to create a synthetic and egocentric dataset for the training of mobile and real-time neural networks. We analyzed the key elements of this synthetic dataset in order to remove the need for real images during training. Then, we proposed to use the depth information contained in RGB-D images to improve the performance of the object detector. We tackled the issue of domain generalization from synthetic to real RGB-D images, and we proposed different approaches in order to reduce the reality gap, that are compatible with a mobile and real-time inference.

Keywords : object detection, deep learning, synthetic dataset, egocentric viewpoint.

Table des matières

Introduction générale

Dans l'industrie, tous les produits ne sont pas fabriqués en quantités massives sur des chaînes d'assemblage automatisées. Certains sont produits en petites quantités, de l'ordre de quelques dizaines ou centaines d'unités, voire encore moins lorsqu'il s'agit d'un prototype. Dans ces cas-là, les risques de commettre une erreur sont plus élevés que lorsque l'assemblage est connu et le geste est répétitif ; la qualité du produit fini en est alors réduite. Une recherche de polyvalence des opérateurs a été rapportée, afin de s'adapter à des assemblages différents et innovants.

L'entreprise DEMS, qui participe à l'encadrement de cette thèse, est un bureau d'étude en ingénierie et design industriel, spécialisé dans la conception de produits liés au domaine des transports. DEMS accompagne ses clients depuis l'idée jusqu'à la conception de véhicules, avec des processus assurant le développement de solutions de mobilité sûres et durables. L'entreprise est ainsi présente tout au long de la chaîne de conception et production, et a une connaissance directe des difficultés liées à l'assemblage de produits dans le cas de petites et moyennes séries.

D'un autre côté, les outils numériques ont récemment été plébiscités dans de nombreux contextes, ce qui a encouragé leur développement. On parle de « transformation digitale » des entreprises, ou « digitalisation », pour désigner la transformation des processus traditionnels par les technologies numériques. Initiée par l'arrivée d'Internet, cette transformation permet de communiquer les informations de façon plus efficace et instantanée, en partageant des documents numériques de tous types très simplement. Dans l'industrie, le numérique a trouvé sa place avec le suivi des produits et commandes, la gestion des stocks, les logiciels métiers (qui répondent aux besoins spécifiques d'une entreprise ou d'un métier [1]), etc. Plus récemment, les nouvelles technologies comme la réalité virtuelle et la réalité augmentée ont également fait irruption dans les usines afin de proposer des formations plus efficaces en temps et en coût aux employés (et ainsi améliorer leur polyvalence), ou encore pour remplacer les nombreux documents imprimés utilisés jusque-là.

Dans ce contexte, l'entreprise DEMS s'est intéressée au sujet du numérique afin de fluidifier les processus et transferts d'information entre le bureau d'étude et l'usine. En particulier, un décalage a été constaté entre l'étape de conception du produit, effectuée via un ou plusieurs logiciels spécifiques et générant des données numériques généralement en 3D, et l'étape d'assemblage de ce produit, reposant sur des processus plus manuels et ne faisant pas usage des outils technologiques à disposition. C'est ainsi que le projet CESAR (*Communication Engineering System of Augmented Reality*) est né, dont le but initial était de proposer aux opérateurs un casque de réalité augmentée qui superposerait à la réalité les informations liées à l'assemblage. Grâce à une ou plusieurs caméras et à un système d'affichage, les étapes d'assemblage pourraient être visualisées directement dans

1. `https://www.industrie-news.com/zoom-sur-le-marche-des-logiciels-metiers.html`, consulté le 25/04/2022

l'espace de travail en utilisant les modèles 3D des pièces, et il serait également possible de signaler à l'opérateur toute erreur en temps réel, avant qu'elle ne puisse plus être corrigée. En parallèle de cette thèse, le projet CESAR a évolué pour devenir Katapuls, un ensemble d'outils numériques pour l'industrie facilitant le partage des données entre DEMS et ses différents collaborateurs. L'offre commerciale a finalement vu le jour en fin d'année 2021 avec AVA [2], un écosystème d'offres et d'outils pour l'accompagnement des projets industriels en valorisant les modélisations 3D tout au long du processus de fabrication d'un produit.

C'est dans ce cadre ambitieux que s'inscrit cette thèse, dont le but est de permettre l'interaction en temps réel entre les outils proposés avec AVA et l'environnement industriel des opérateurs lors de l'assemblage. Elle s'est déroulée en collaboration avec l'équipe IMAGINE du laboratoire LIRIS, spécialiste de la recherche en vision par ordinateur, apprentissage et reconnaissance de formes. Nous avons abordé ces thématiques, et en particulier le traitement des images de l'environnement de l'opérateur afin d'y identifier les objets d'intérêt, dont les modélisations 3D proviennent de l'étape de conception.

Cette thèse se compose de 5 chapitres :

- Le Chapitre 1 présente les notions liées au contexte industriel et les problématiques rencontrées. Un état des lieux des entreprises et produits commerciaux liés à la réalité augmentée est également proposé.
- Le Chapitre 2 correspond à une revue de la littérature des différents thèmes scientifiques abordés : le traitement d'images par apprentissage profond, les réseaux de neurones artificiels utilisés pour la détection d'objets, l'utilisation des modèles 3D, ainsi que les images égocentriques. Les notions nécessaires à la compréhension de ce manuscrit y sont définies.
- Le Chapitre 3 introduit nos travaux autour de la génération d'images synthétiques égocentriques pour l'apprentissage profond en utilisant uniquement des modèles 3D.
- Le Chapitre 4 décrit l'étude que nous avons menées sur deux détecteurs d'objets entraînés avec des images synthétiques à trois canaux de couleur (appelées « images RGB », pour *Red*, *Green*, *Blue*).
- Le Chapitre 5 expose les méthodes que nous avons proposées pour intégrer une seconde modalité d'image afin de mieux traduire la géométrie d'une scène, et ainsi améliorer la reconnaissance des objets industriels.

Ces travaux ont fait l'objet de plusieurs publications dans des congrès scientifiques. Notre méthode de génération d'images synthétiques égocentriques et les résultats de détection associés ont été publiés lors de la conférence internationale VISAPP 2020 (*International Conference on Computer Vision Theory and Applications*) (Cohen *et al.*, 2020). Cet article a reçu le prix du « Meilleur Article Industriel ». Une partie de notre analyse de la détection d'objets industriels avec un réseau de neurones mobile a été publiée lors de la conférence internationale ICIP 2021 (*International Conference on Image Processing*) (Cohen *et al.*, 2021a) et lors de la conférence nationale ORASIS 2021 (journées francophones des jeunes chercheurs en vision par ordinateur) (Cohen *et al.*, 2021b). Enfin, nos travaux liés à la détection d'objets industriels à partir d'images multimodales sont en cours de soumission pour publication.

2. `https://www.ava-ux.com/`

Chapitre 1

Contexte industriel

Dans ce premier chapitre, nous décrivons le contexte industriel dans lequel s'inscrit cette thèse. Nous présentons d'abord ce qu'est la réalité augmentée et comment cette technologie s'insère dans le paysage industriel. Ensuite, nous décrivons les données disponibles dans cette thèse, et les objectifs et contraintes associées au projet de l'entreprise DEMS. Enfin, nous clarifions notre positionnement par rapport à l'état de l'art et aux objectifs fixés.

Sommaire

1.1 La réalité augmentée dans l'industrie

1.1.1 La conception d'un produit industriel

Le processus de conception d'un produit industriel complexe tel qu'un bus électrique se compose de différentes étapes. La première correspond à la définition du besoin par le client, à partir duquel un modèle 2D ou 3D est conçu avec un logiciel de modélisation. On l'appelle « modèle CAO » pour « Conception Assistée par Ordinateur », ou CAD en anglais, pour « Computer-Aided Design ». Ce modèle comprend tous les éléments qui composent le produit final, incluant les pièces standards (comme les vis ou les écrous) et les pièces à fabriquer spécifiquement. Lorsque la CAO du produit est validée, vient l'étape de mise en plan, qui décrit chacune des pièces qui composent le produit final. Chaque élément est dessiné en 2D avec toutes les informations et références nécessaires afin d'être fabriqué (dimensions, processus d'usinage, finitions, etc.). Pour un ensemble de pièces, le guide opératoire indique ensuite la façon dont ces pièces doivent être assemblées, étape par étape, en général sous forme de schémas annotés. Il sert aux employés des usines d'assemblage, qui ont pour mission de fabriquer un prototype ou le produit final. Cette étape est donc cruciale pour assurer la qualité du produit fini ; elle doit comporter toutes les informations nécessaires et dans le bon ordre. Ainsi, les informations liées aux modèles 3D sont finalement utilisées sous forme 2D, mises « à plat » pour être consultées lors de cette étape d'assemblage. La création de ce guide opératoire est très longue et peut difficilement être automatisée ; et bien qu'il soit important, il n'apporte pas de valeur au produit.

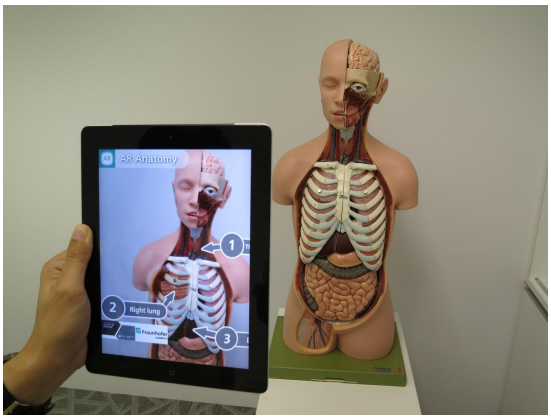
Grâce à l'essor des outils numériques, les modèles 3D sont aujourd'hui utilisés pour bien plus que la simple conception des pièces, et sont ainsi valorisés tout au long de la production. Visualisés lors des revues de projet grâce à la réalité virtuelle, les modèles 3D sont également utilisés pour produire un guide opératoire virtuel, qui permet de remplacer le format papier et de faciliter l'interaction. En effet, les différentes étapes d'assemblage peuvent être représentées dans des scènes virtuelles sur un écran d'ordinateur, de *smartphone* ou de tablette tactile, ou même avec des technologies plus immersives. Ainsi, les opérateurs ont la possibilité de déplacer les pièces ou les faire tourner, de les agrandir ou réduire, ou encore d'afficher et masquer des informations afin de diminuer le nombre d'éléments visibles. L'utilisation des données numériques dans les usines est un sujet d'actualité qui permet de réduire les documents imprimés, et de simplifier le transfert d'informations entre le bureau d'études et l'usine.

1.1.2 La réalité augmentée

La Réalité Augmentée (RA) est une technique d'affichage d'informations numériques qui superpose des éléments virtuels au monde réel. Souvent confondue avec la Réalité Virtuelle (RV), elle correspond à l'opposé du spectre formé par la Réalité Mixte (RM). La RM correspond à l'ensemble des technologies mêlant éléments du monde réel et éléments virtuels, ou numériques. En RV, une personne est totalement immergée dans un environnement virtuel, comme le montre la Figure 1.1a ; elle peut ainsi se retrouver à l'intérieur d'un jeu vidéo, tout comme dans une salle de réunion avec des collègues physiquement présents à l'autre bout du monde. Les utilisateurs sont totalement coupés du monde réel, au contraire de la RA qui introduit des éléments virtuels dans un environnement réel comme sur la Figure 1.1b, qui ajoute des indications virtuelles sur l'écran d'une tablette.

Les différents supports de RA, qui sont décrits dans la suite, ont le point commun de laisser l'utilisateur apercevoir le monde réel qui l'entoure, soit directement à travers des verres de lunettes transparents, soit à travers un écran occultant mais montrant l'environnement réel grâce à une caméra. La RA permet d'interagir avec les objets du monde réel, par exemple en affichant des informations liées à des objets qui entourent l'utilisateur ou en affichant de nouveaux objets en taille réelle. Ces deux technologies ont un large panel d'applications, du divertissement à l'industrie en passant par le milieu médical ou l'éducation. Azuma (1997) résume dans un rapport exhaustif les connaissances autour de la RA dans les années 90, incluant les différentes technologies et les méthodes de traitement de données associées. Plus récemment, Billinghurst *et al.* (2015) actualisent ce travail avec les avancées du début des années 2000, tout comme Manuri et Sanna (2016) de façon beaucoup plus concise. La revue de l'état de l'art de Li *et al.* (2017) se concentre sur l'application de la RA au domaine de l'ingénierie, en particulier les méthodes qui visent à replacer dans le monde réel les résultats d'analyses et de simulations pour en avoir une meilleure compréhension. Dans cette thèse, nous nous intéressons à l'utilisation de la RA comme outil de visualisation de données dans l'industrie.

(a) Casque de réalité virtuelle.



(b) Réalité augmentée à travers une tablette.

FIGURE 1.1 – Exemples de supports pour la RV et RA.

La RV nécessite généralement un appareil appelé « casque RV », constitué d'un écran placé devant les yeux de l'utilisateur, et éventuellement de haut-parleurs. Si les casques ont beaucoup évolué les dernières années et leurs prix sont devenus plus accessibles, ils restent encore assez lourds et inconfortables, ce qui empêche leur utilisation pendant une période prolongée. De plus, ils sont souvent reliés par un câble à une source d'alimentation et à un ordinateur, limitant les mouvements de l'utilisateur. En revanche, la qualité des écrans s'est fortement améliorée, réduisant les effets de « *motion sickness* », une forme de « mal des transports » causant vertiges et nausées chez certains utilisateurs. Il est toutefois déconseillé de porter un casque RV pendant une période prolongée, notamment afin d'éviter les troubles de la vision ou une perte d'orientation, les effets à long terme n'étant pas encore connus.

Pour la RA, les supports possibles sont plus nombreux, mais plus difficiles à intégrer à la vie quotidienne. Le support le plus simple pour la RA est le *smartphone* ; les nombreux « filtres » apparus sur les applications de réseaux sociaux en témoignent. En effet, l'ajout ou la modification d'éléments sur une photo ou une vidéo correspondent à de la RA, et sont aujourd'hui très faciles à créer. Plusieurs jeux vidéo disponibles sur *smartphones*

utilisent également la RA pour faire du monde réel un terrain de jeu, en utilisant la localisation de l'utilisateur et les éléments de l'environnement comme déclencheurs afin de faire apparaître des éléments virtuels à travers l'écran. Cette technique de RA est connue sous le nom de *video see-through*, par opposition à la technique *optical see-through*. Des casques de RA *video see-through* existent et sont très similaires aux casques de RV, mais il est à nouveau déconseillé de les porter pendant une durée trop importante. Au contraire, la RA *optical see-through* repose sur des lunettes (ou un casque) dont les verres sont transparents et sur lesquels sont projetés les objets virtuels. Ainsi, l'utilisateur observe les éléments réels et virtuels de la même façon, dans l'environnement directement, et non à travers un écran. Cette technologie n'est cependant pas encore au point pour une utilisation pour le grand public, en raison de son prix très élevé et de la difficulté à rendre ce support ergonomique. En effet, chaque verre doit être associé à un projecteur, lui-même connecté à un ordinateur qui génère les images, ce qui rend l'ensemble assez lourd. De plus, ces lunettes nécessitent une ou plusieurs caméras afin de proposer du contenu virtuel adapté à l'environnement, de façon *intelligente*. Une autre approche consiste à afficher des informations qui ne dépendent pas de l'environnement, afin d'éliminer le besoin d'une caméra. Sans caméra, les possibilités sont cependant assez limitées, ce qui encourage de nombreuses entreprises à en intégrer au moins une, couplée à un algorithme de traitement d'images. Cela permet de n'afficher une information que lorsqu'elle est pertinente, comme lorsqu'un objet est détecté par exemple. Dans le domaine industriel, une approche de RA plus courante est l'utilisation de systèmes de projection, qui affichent des informations directement sur les surfaces (Cortes *et al.*, 2018). Cette solution pose cependant la question de la confidentialité : tout le monde peut voir les informations projetées, ce qui peut représenter un problème de sécurité.

1.1.3 La réalité augmentée dans l'industrie et la recherche

Au début de cette thèse, nous avons recensé les différentes solutions de RA proposées dans le commerce, à destination du grand public et de l'industrie. Nous avons identifié une quarantaine d'acteurs spécialisés dans le développement de logiciels dédiés à la RA, des technologies de projection d'éléments virtuels, et des casques ou lunettes de RA complets. Parmi ces entreprises, la plupart ne proposaient pas encore de produit prêt à être commercialisé, et ne sont malheureusement pas arrivées jusque-là. Si la RA suscite beaucoup de réactions et de grandes attentes, le peu d'engouement lors de la commercialisation des appareils témoigne du manque de maturité de la technologie, présentant de nombreuses limitations pour une utilisation par le grand public. En conséquence, la plupart des entreprises du secteur ont fait faillite depuis 2018 ou abandonné le secteur de la RA, laissant à seulement quelques acteurs importants le soin de développer leurs produits. Ces acteurs sont de deux types : une grande entreprise pour laquelle la RA ne représente qu'une petite partie des activités, et qui se place comme précurseur en attendant une évolution favorable du marché pour le grand public ; une entreprise spécialisée dans une application de RA extrêmement spécifique.

Les casques et lunettes de RA qui ont été commercialisés sont généralement à destination des entreprises ou activités commerciales. Par exemple, Epson [3] s'est placé sur le segment de la culture avec un modèle de lunettes RA pratique et ergonomique pour améliorer l'expérience des visiteurs dans les musées ; ils ont également proposé une ver-

3. `https://www.epson.fr/fr_FR/produits/casque-de-réalité-augmentée/c/smart-glasses`

sion plus robuste de ces lunettes pour les ouvriers sur les chantiers. Les Google Glass [4], initialement développées pour le grand public, ont finalement trouvé leur place dans les usines également, pour la collaboration à distance notamment. Les lunettes intelligentes à destination des sportifs ont également eu du succès, à l'instar des SOLOS de l'entreprise américaine Kopin ou des Raptor de l'entreprise israélienne Everysight. Les lunettes proposées pour le grand public sont plutôt des extensions du *smartphone*, comme les Vuzix Blade (Figure 1.2a). Certains des modèles mentionnés précédemment ne comportent pas de caméra, et servent uniquement à afficher de l'information sans que l'utilisateur n'ait besoin de détourner son regard. L'interaction se fait par contrôle vocal, par boutons (tactiles ou non) sur les branches des lunettes, ou par le *smartphone* directement. Les fonctionnalités limitées de ces appareils leur permettent cependant d'avoir une durée de batterie suffisante pour les applications concernées, avec un poids presque équivalent à celui d'une paire de lunettes classique.

Parmi tous les casques de réalité augmentée, le plus connu et abouti est celui de Microsoft, appelé HoloLens (Figure 1.2b). Bien qu'il soit utilisé par des grandes entreprises pour de nombreuses applications industrielles, comme l'assemblage et la maintenance, il n'est pas encore adopté par les petites et moyennes entreprises, notamment en raison de son prix trop élevé. Du côté des laboratoires de recherche, plusieurs travaux étudient les capacités de ce casque dans le cadre d'une tâche d'assemblage (Evans *et al.*, 2017), ou encore du suivi et de la détection d'objets en temps réel (Farasin *et al.*, 2020). Park *et al.* (2021) résument 44 publications scientifiques analysant les performances de ce casque de RA utilisé dans différents contextes et applications.

(a) Vuzix Blade [5].

(b) Casque HoloLens de Microsoft [6].

FIGURE 1.2 – Exemples de lunettes et casque de réalité augmentée.

D'un point de vue logiciel, la RA aborde de nombreux problèmes de traitement d'images. L'estimation de pose correspond à la localisation de la caméra dans l'environnement, en mesurant le mouvement d'une image à l'autre d'une séquence vidéo (Marchand *et al.*, 2016). Cela peut aussi revenir à estimer la position d'un objet connu dans une image. Cette tâche est particulièrement importante lorsque la caméra est en mouvement, afin que l'objet virtuel projeté dans l'environnement apparaisse toujours au même endroit

4. https://www.google.com/glass/start/
5. https://www.vuzix.com/products/vuzix-blade-smart-glasses-upgraded
6. https://www.microsoft.com/fr-fr/hololens

et de façon « lisse », sans sursauts ni délai. Les algorithmes de SLAM (*Simultaneous Localization and Mapping*) visent à localiser l'utilisateur tout en cartographiant son environnement (Davison *et al.*, 2007; Klein et Murray, 2007; Mur-Artal *et al.*, 2015). En particulier, les méthodes vSLAM (pour *visual* SLAM) se basent uniquement sur des images, sans avoir recours à d'autre types de capteurs (Taketomi *et al.*, 2017). Une approche classique pour la localisation est l'utilisation de « marqueurs », des éléments 2D dont le patron numérique est connu à l'avance afin de le retrouver dans l'environnement. Lorsqu'ils sont détectés, ils servent de point d'ancrage pour placer un élément virtuel (Pombo et Marques, 2017). Ces marqueurs peuvent être artificiels, tels que des schémas ou motifs générés par ordinateur (codes QR, logos, etc.) ; ils sont généralement utilisés pour placer un objet virtuel à un endroit précis en imprimant le motif sur un support ou une feuille de papier. La Figure 1.3 montre quelques exemples de marqueurs artificiels : un code QR, et un cube dont chaque face est repérée par un schéma différent. Des marqueurs dits « naturels » peuvent également être reconnus dans l'environnement. Ce sont principalement des profils urbains ou naturels, comme une ligne d'horizon avec des immeubles reconnaissables, ou une chaîne de montagne. Ces marqueurs naturels sont choisis pour leur caractère immuable, ils ne seront pas modifiés dans le temps (ou à très long terme), et peuvent donc servir à localiser un utilisateur dans un lieu extérieur (Ayadi *et al.*, 2018). Dans l'industrie, il est en général plus simple de fixer une caméra sur un plan de travail, et de définir à l'avance les zones d'intérêt, comme des bacs de composants ou des stations pour poser les outils. Un avantage est de ne pas avoir besoin de localiser ces composants ou outils en temps réel ; les inconvénients sont la nécessité de préparer l'espace de travail, de respecter ces zones, et de les redéfinir pour chaque nouvel espace de travail et projet. Enfin, la notion de suivi temporel (*tracking*) revient souvent dans les travaux liés à la RA. Lorsque l'environnement est cartographié, les zones qui ont été observées une fois sont mémorisées puis mises à jour si nécessaire. Lorsqu'un objet est identifié, il est suivi d'une image à l'autre du flux vidéo, dans un simple objectif de localisation, ou bien dans le but d'identifier un évènement, un changement d'état ou une action particulière de l'utilisateur (Runz *et al.*, 2018). Le suivi est également fait de façon plus rapide que la détection de l'objet, ce qui permet d'avoir un système plus rapide que si la détection était répétée dans chaque image. Les algorithmes de suivi peuvent cependant souffrir d'une dérive dans le temps, surtout si les objets sont parfois occultés, ce qui demande de les recalibrer régulièrement ou de ne pas les utiliser seuls.

(a) Code QR[7].

(b) « merge Cube »[8].

FIGURE 1.3 – Exemples de marqueurs artificiels pour la réalité augmentée.

7. https://fr.wikipedia.org/wiki/Code_QR
8. https://mergeedu.com/cube

Toutes ces tâches peuvent s'appuyer sur un ou plusieurs modèles 2D ou 3D des objets à identifier, ou bien elles identifient des objets remarquables dans la scène (Bleser *et al.*, 2005). Elles peuvent également utiliser un mélange des deux approches. Les algorithmes de SLAM identifient généralement des points caractéristiques dans l'environnement, et s'en servent pour recaler les images les unes par rapport aux autres. Elles peuvent aussi s'appuyer sur l'identification d'un objet dont le modèle est connu (Tamaazousti *et al.*, 2011). Les méthodes proposées dans la littérature s'appuient cependant souvent sur des contraintes fortes limitant le type d'objets pouvant être utilisés, comme la géométrie (surfaces planes ou d'arêtes) ou la texture de ces objets (Hodan *et al.*, 2015).

De nombreux points sont encore bloquants pour le déploiement de technologies de RA dans le monde réel. La variabilité dans les technologies et le peu de supports effectivement commercialisés rendent difficiles les études utilisateur, qui analysent par exemple les méthodes de manipulation d'objets virtuels (He et Yang, 2014) ou l'intégration réaliste des éléments virtuels (Klein et Murray, 2009). Quelques travaux étudient l'utilisation de casques de RA pour assister les opérateurs lors de tâches d'assemblage (Besbes *et al.*, 2012; Blattgerste *et al.*, 2017), mais l'évaluation même d'un système de RA est encore en cours de discussion (Funk *et al.*, 2015). Une étude en situation réelle a montré une amélioration des performances pour les opérateurs peu expérimentés, mais une diminution de la performance pour les opérateurs experts, lors de l'utilisation d'instructions d'assemblage en RA (Funk *et al.*, 2017). De façon générale, les études menées ne comportent qu'une dizaine ou vingtaine d'utilisateurs à chaque fois, un faible échantillon pour évaluer l'impact de la RA si elle venait à être déployée à grande échelle ; de plus, elles ne montrent que des bénéfices limités à utiliser des dispositifs de RA (Büttner *et al.*, 2020).

Plusieurs exemples encourageants d'utilisation de la RA dans des contextes industriels ont toutefois été proposés. Gay-Bellile *et al.* (2012) présentent une application de maintenance d'un moteur de voiture assistée par RA ainsi qu'une application de personnalisation de l'apparence d'une voiture, basées sur leur méthode de SLAM contraint par un modèle 3D (Tamaazousti *et al.*, 2011). En s'appuyant sur le même principe, Bourgeois *et al.* (2016) présentent une solution complète pour suivre un modèle 3D d'un objet en temps réel et l'annoter afin que les informations apparaissent en RA, en seulement quelques minutes. Leur système de visualisation est une tablette industrielle, ergonomique et robuste, qui permet l'ajout d'éléments virtuels directement sur l'objet d'intérêt. Ces travaux ont mené à la création de l'entreprise Diota [9], aujourd'hui une référence dans les solutions digitales pour l'industrie.

1.2 Objectifs et données de la thèse

1.2.1 Valorisation de la 3D avec la RA

Dans un objectif de réutiliser au maximum les éléments 3D créés lors de la phase de conception des produits, ainsi que de simplifier la mise en plan et l'assemblage, DEMS souhaite créer un guide d'assemblage intelligent en réalité augmentée. Les opérateurs doivent pouvoir visualiser les différentes étapes tout en ayant les mains libres pour manipuler les pièces ; ces pièces doivent également être identifiées et localisées dans l'environnement, afin d'y superposer leurs modèles CAO ou les informations associées. Dans un second temps, ce système de RA pourra comparer précisément les objets réels et le guide d'assemblage

9. `https://www.diota.com/`

virtuel afin de détecter les différences, et ainsi prévenir l'opérateur d'une possible erreur avant qu'elle ne soit irréparable. Ces deux fonctionnalités reposent sur une même problématique : détecter les objets réels dans l'environnement de l'utilisateur. Dans l'optique de réduire au maximum les interventions humaines, l'acquisition et l'annotation d'images représentant les pièces à reconnaître sont impossibles. En revanche, les modèles 3D sont disponibles.

1.2.2 Solution adoptée

Dans notre application, nous n'avons pas accès aux espaces de travail avant l'assemblage des produits, ce qui empêche l'installation d'une caméra fixe et de marqueurs pour placer les objets. De plus, nous voulons afficher des informations virtuelles visuellement proches des objets détectés, et non à une position définie par rapport à un marqueur fixe. Après une étude approfondie des solutions de RA existantes, nous avons fait les choix de conception suivants pour valoriser les modèles 3D :

- pour le matériel, un casque de RA *optical see-through* intégrant une ou plusieurs caméras ;
- pour le logiciel, une solution permettant la reconnaissance et la localisation des pièces manipulées par l'opérateur portant le casque.

En ce qui concerne le casque de RA, DEMS a choisi de concevoir son propre appareil, ce qui présente les avantages suivants : pas de dépendance à une entreprise externe (et à un produit qui peut subitement ne plus être commercialisé, ou dont les caractéristiques techniques peuvent évoluer), le choix des caméras et capteurs est libre. En particulier, un choix de conception de ce casque RA est de le rendre totalement mobile, en l'associant à un ordinateur embarqué porté par l'opérateur. La conception des algorithmes de traitement d'images est donc étroitement liée à la conception du casque, puisque les choix matériels tels que l'ordinateur embarqué définissent la capacité de calcul disponible. A partir de ce choix, nous avons défini les contraintes logicielles associées à cette thèse.

Une première contrainte importante est l'absence de GPU (*Graphical Processing Unit*). En effet, ce composant spécialisé dans le traitement des images est volumineux et consomme une quantité importante d'énergie. Il a donc une très grande influence sur la conception finale du système, et sur son autonomie, qui doit être suffisante pour une période de travail (soit environ quatre heures). Bien que les algorithmes de traitement d'images s'appuient souvent sur un GPU, notamment lors de l'utilisation de méthodes d'apprentissage machine, le cahier des charges défini en début de thèse demandait que l'algorithme de reconnaissance des objets puisse s'appliquer en temps réel (ce qui correspond en général à 30 images par secondes) sans GPU. Ensuite, l'algorithme proposé doit pouvoir s'adapter à chaque projet, c'est-à-dire chaque nouvel assemblage de produit, peu importe la taille ou la géométrie du produit, mais également à chaque usine dans laquelle il doit être assemblé. Lorsque les modèles 3D des pièces composant un produit sont définitivement validées, le logiciel spécifique à ce produit doit être créé automatiquement, et fonctionner sans intervention d'un expert en informatique. La méthode développée doit donc être autonome et ne pas demander d'intervention humaine. Il est enfin à noter que le placement de la caméra donne un point de vue particulier aux images acquises et traitées, appelé point de vue « égocentrique », ou « à la première personne », qui se différencie notamment des images classiques par le mouvement permanent de la caméra.

1.2.3 Cas d'usage

Tout au long de cette thèse, nous avons appliqué nos travaux à un cas particulier, directement tiré du milieu industriel. Ce cas d'usage est l'assemblage d'un siège de bus commercialisé par l'entreprise RUSPA, conçu par DEMS. Disposant des modèles 3D correspondant à cinq pièces de ce siège, ainsi que des prototypes réels de ces pièces, nous avons pu étudier différentes approches afin de localiser ces pièces dans des images réelles, en utilisant uniquement les modèles 3D comme source d'information. Nous appelons « RUSPA » ce jeu de données.

(a) Images acquises avec un *smartphone* Honor 6C Pro.

(b) Images acquises avec un *smartphone* iPhone 7 Plus.

FIGURE 1.4 – Images de RUSPA issues de vidéos RGB égocentriques acquises avec un support Aryzon [10] au début de la thèse.

En parallèle de cette thèse, DEMS s'est attelé à la conception du casque de RA. Nous avons commencé par acquérir des images égocentriques avec des supports de RA pour *smartphones* disponibles dans le commerce et à bas coût. Quelques images acquises avec deux appareils différents sont montrées sur la Figure 1.4. Ensuite, nous avons utilisé une caméra RGB-D Intel RealSense D435i fixée sur un prototype de casque de RA de DEMS afin d'acquérir de nouvelles images égocentriques, avec à la fois une images RGB et une carte de profondeur ; quelques exemples sont représentés sur la Figure 1.5. Enfin, les modèles 3D des pièces du siège de bus sont représentées sur la Figure 1.6. L'annexe B contient d'autres images acquises avec ces supports.

L'analyse des images réelles nous a permis de définir un certain nombre d'éléments clés pour la suite, notamment les difficultés inhérentes aux images égocentriques et à l'application d'assemblage d'un produit industriel. Tout d'abord, la caméra placée au niveau de la tête offre un champ de vision très réduit, variant légèrement d'un support testé à l'autre. En effet, la caméra étant très proche de la scène d'intérêt, la scène visible dans les images est réduite par rapport à la scène vue par l'utilisateur. En particulier, lorsque

10. `https://www.aryzon.com/`

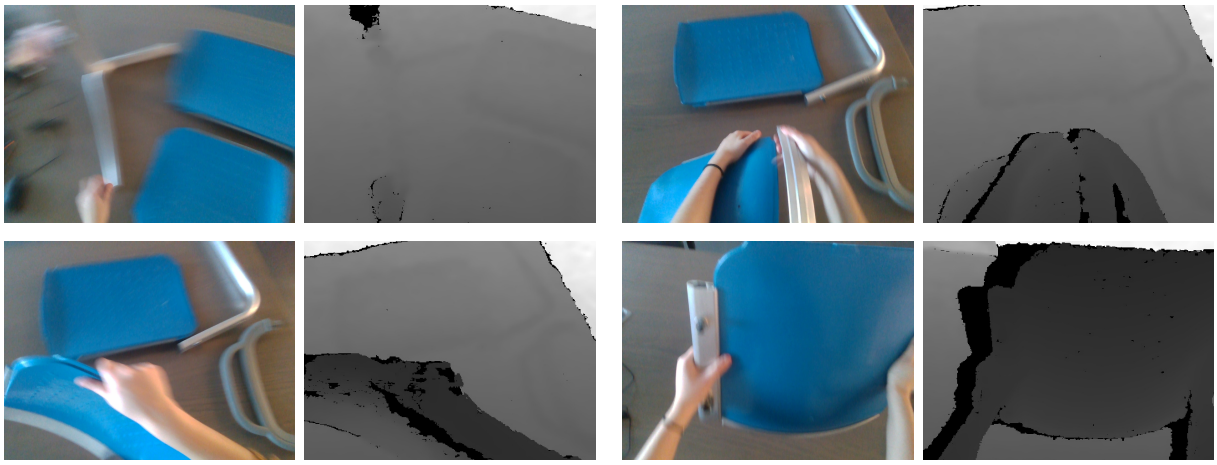
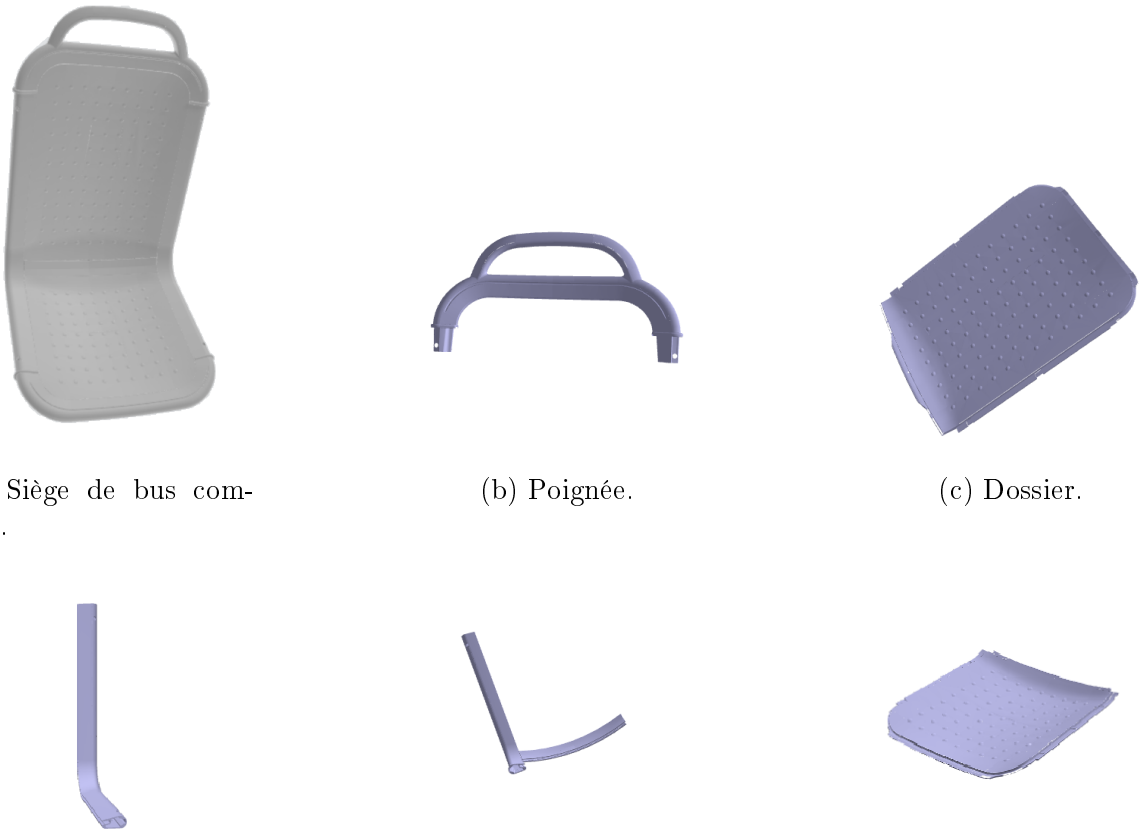


FIGURE 1.5 – Images RGB-D de RUSPA extraites de vidéos égocentriques acquises par la caméra Intel RealSense D435i et le prototype de casque de réalité augmentée de DEMS.



(a) Siège de bus complet. (b) Poignée. (c) Dossier.

(d) Profil droit. (e) Profil gauche. (f) Assise.

FIGURE 1.6 – Les modèles 3D associés au siège de bus RUSPA. (a) siège assemblé, (b)-(f) les différents objets à détecter.

les objets manipulés sont de taille assez grande, comme les éléments du siège RUSPA, ils sont presque toujours partiellement visibles. Ces images sont donc très différentes de ce que montrent les solutions de RA du commerce, qui utilisent souvent une tablette comme support d'acquisition et de visualisation, ce qui permet de s'éloigner du plan de travail.

En particulier, si la caméra n'est pas bien orientée, l'opérateur doit « montrer » les objets à la caméra, avec une gestuelle qui n'a rien de naturel. Cependant, même en surélevant les objets ou en penchant la tête, les objets n'apparaissent généralement pas entièrement dans les images, ce qui représente une limitation pour de nombreux algorithmes de reconnaissance d'objets, qui nécessitent de voir l'objet en entier. Les mouvements des bras lors de l'assemblage ajoutent également des occultations, ainsi que des ombres sur les objets et l'environnement. Enfin, la caméra étant en mouvement avec la tête de l'utilisateur, et les objets ayant aussi un mouvement propre lorsqu'ils sont manipulés, ces derniers apparaissent très souvent flous. Il est donc très difficile d'associer les objets dans ces images à des modèles 3D avec les techniques habituelles utilisées en RA.

1.3 Positionnement de la thèse

Comme décrit précédemment, le projet Katapuls a pour but de valoriser les données de conception 3D et de faciliter la transmission d'informations grâce aux outils numériques. Les trois axes principaux de ce projet sont la conception d'un casque de RA, le développement d'un guide opératoire numérique, et l'intégration d'une solution d'intelligence artificielle afin que les éléments virtuels s'adaptent automatiquement à l'environnement, au contexte et aux objets présents. L'objectif de cette thèse est d'identifier les objets industriels présents dans les images égocentriques, telles que celles issues d'un casque de RA, uniquement à partir des modèles 3D de ces objets. L'algorithme proposé doit fonctionner sans intervention humaine, et doit donc s'adapter à une large variété de pièces et de projets, sans jamais en voir d'images réelles.

Nous avons vu que les images obtenues avec un casque de RA présentent des difficultés liées à leur point de vue égocentrique. En particulier, l'apparence des objets varie fortement, avec des vues tronquées et des changements d'illumination. Dans ces conditions, l'apprentissage profond a montré de meilleurs résultats que l'apprentissage classique : en fournissant un très grand nombre de données d'entraînement, le modèle est capable de faire abstraction des détails non pertinents pour apprendre à reconnaître les objets. Cependant, le choix d'utiliser des méthodes d'apprentissage profond apporte d'autres contraintes. Tout d'abord, un grand nombre de données est nécessaire ; n'ayant à disposition que les modèles 3D des objets à reconnaître, nous avons exploré les différentes possibilités afin de les utiliser sans aucune image réelle. Ensuite, les modèles d'apprentissage profond sont connus pour être volumineux et lents, difficiles à embarquer dans une application mobile. Nous nous sommes donc concentrés sur les solutions respectant ces contraintes d'efficacité. Enfin, l'apprentissage profond est aujourd'hui considéré pour résoudre un très grand nombre de problèmes de traitement d'images. Nous avons choisi la tâche de détection d'objets dans des images pour plusieurs raisons :

- C'est une tâche suffisamment connue pour nous permettre de nous concentrer sur les aspects de l'entraînement avec des modèles 3D uniquement et du point de vue égocentrique. Une fois ces difficultés maîtrisées, il sera possible d'étendre les connaissances obtenues à d'autres tâches, telles que l'estimation de pose.
- L'annotation d'images réelles pour la détection 2D est plus simple et rapide que pour d'autres tâches (comme l'estimation de pose ou le suivi temporel). Nous avons donc pu acquérir et annoter plusieurs centaines d'images pour évaluer les solutions proposées avec le jeu de données RUSPA, notre cas d'usage.

En résumé, nous avons appliqué des méthodes d'apprentissage profond à la détection

d'objets industriels dans des images égocentriques, à partir des modèles 3D des objets d'intérêt. Dans un premier temps, nous avons généré des images synthétiques RGB pour l'entraînement d'un premier détecteur d'objets de petite taille. Ensuite, nous avons ajouté la notion de profondeur avec des images RGB-D. Nous avons pour cela généré des images synthétiques RGB-D, toujours de façon automatique à partir des modèles 3D. Les deux méthodes de génération d'images sont présentées dans le Chapitre 3. Nous avons ensuite étudié la détection d'objets dans des images RGB égocentriques dans le Chapitre 4, et dans les images RGB-D dans le Chapitre 5. En particulier, nous y abordons les problématiques de la fusion de modalités et de la généralisation de l'apprentissage des images synthétiques vers les images réelles.

Chapitre 2

Etat de l'art

Ce chapitre présente l'état de l'art des différentes thématiques abordées dans cette thèse. Après une présentation générale de l'apprentissage profond, des bases théoriques aux différentes phases d'apprentissage, nous détaillons les architectures spécifiques les plus populaires et les plus appropriées pour notre application. Nous présentons ensuite un panorama des différentes façons d'utiliser un modèle 3D d'un objet à reconnaître, puis les travaux en lien avec les images égocentriques.

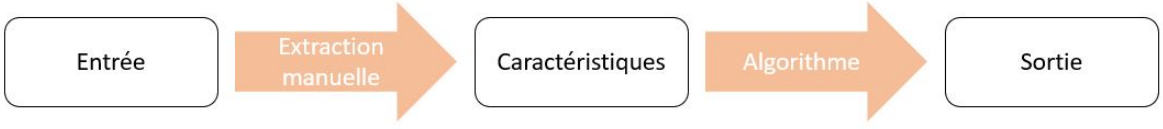
Sommaire

2.1 Notions de base

2.1.1 Traitement d'images par ordinateur

Le traitement d'images par ordinateur est un champ d'application de l'informatique consistant à manipuler des images numériques, à des fins d'amélioration de la qualité ou d'extraction d'information. Améliorer la qualité peut revenir à réduire le bruit de l'image, coloriser une image initialement en noir et blanc, ou encore combler des zones manquantes. Les informations qu'il est possible d'extraire sont de nature différentes et dépendent souvent du type de document concerné : extraire le texte d'un document manuscrit, identifier le type de scène représenté dans une image (par exemple, un paysage ou une scène urbaine), identifier le type d'œuvre (une photographie ou une peinture, et éventuellement l'artiste), ou encore analyser les éléments contenus dans cette image (compter le nombre de personnes, détecter un bagage abandonné, suivre la trajectoire d'un objet ou d'une personne, etc.).

Les applications sont nombreuses, et les solutions pour y arriver le sont tout autant : classification, segmentation, suivi temporel, etc. Pour chacune de ces approches, les algorithmes proposés évoluent avec le temps et les capacités grandissantes des ordinateurs. En particulier les GPU (*Graphical Processign Units*) sont des processeurs initialement conçus pour le traitement des images et des données visuelles ; l'augmentation de leurs capacités de calcul et de stockage des données ont joué un grand rôle dans le développement des algorithmes d'apprentissage profond. Aujourd'hui, beaucoup de ces tâches sont évaluées en utilisant des *benchmarks*, des jeux de données publics permettant de comparer les algorithmes entre eux. Depuis l'avènement de l'apprentissage profond en 2012 avec le réseau de neurones artificiels AlexNet (Krizhevsky *et al.*, 2012), ce sont ces réseaux de neurones qui permettent d'obtenir très souvent les meilleurs résultats (He *et al.*, 2015). Depuis, les méthodes de traitement d'images sont décomposées en deux grandes familles : les méthodes dites « classiques », et les méthodes basées sur de l'apprentissage profond (*deep learning*). Elles sont schématisées sur la Figure 2.1. Les méthodes classiques se décomposent en plusieurs étapes : nettoyage des données, analyse pour en extraire des caractéristiques, puis application d'un algorithme de classification sur ces caractéristiques. Ces étapes qualifiées de « manuelles » demandent une expertise du domaine d'application ou une bonne compréhension du contexte, afin d'extraire des caractéristiques cohérentes



(a) Méthode classique.

(b) Méthode avec de l'apprentissage profond.

FIGURE 2.1 – Schémas représentant les étapes d'un algorithme de traitement d'images par méthode classique et avec de l'apprentissage profond.

et de choisir le bon algorithme. Au contraire, les méthodes par apprentissage profond s'appuient sur la modélisation de données par l'ajustement des paramètres d'un modèle représentant la relation entre des entrées et des sorties, sans s'attacher à comprendre le fonctionnement précis du système considéré (O'Mahony *et al.*, 2019). Les paramètres sont raffinés par itérations successives afin que le modèle s'ajuste au mieux aux données fournies. Ce modèle est appelé un réseau de neurones artificiels. En général, ces méthodes sont dites « de bout en bout » car l'extraction de caractéristiques est implicite, il n'y a donc qu'une étape entre l'image d'entrée et la sortie.

L'apprentissage profond, comme d'autres algorithmes de l'apprentissage automatique (ou *machine learning*), peut être supervisé ou non. Dans le cas supervisé, les données utilisées pour l'apprentissage sont des paires (entrée, sortie) : la sortie attendue pour chaque image en entrée est fournie à l'algorithme. On appelle « vérité-terrain » ces sorties attendues, ou « *ground-truth* » en anglais. La vérité-terrain est obtenue par un processus d'annotation, souvent manuel : pour chaque entrée, une personne indique ce qu'elle estime être la sortie correcte. Dans le cas non supervisé, l'algorithme identifie des schémas dans les données d'entrée sans être guidé par une sortie attendue, par exemple en regroupant des données similaires (*clustering*, *k-means*, etc.). Dans les deux cas, la qualité du modèle dépend fortement des données fournies pendant la phase d'apprentissage, ou d'entraînement.

Extrêmement efficaces pour extraire et prédire de l'information de nombreux types de données (images, mais aussi audio ou texte), les réseaux de neurones artificiels sont dépendants de la qualité et de la quantité des données fournies à l'entraînement ; ils s'adaptent difficilement lorsque la distribution des données n'est plus la même que celle des données d'entraînement. Bien que leurs paramètres soient connus, ils sont rarement interprétables, ce qui leur vaut le qualificatif de « boîte noire ». Pour résumer, leur grand nombre de paramètres offre une capacité d'apprentissage supérieure aux méthodes classiques, mais requiert également une quantité plus importante de données, pas toujours disponibles. Des méthodes hybrides ont finalement vu le jour, combinant de l'apprentissage profond et des méthodes de traitement d'images classiques, afin de tirer parti des avantages des deux approches (Wojke *et al.*, 2017; Zheng *et al.*, 2018).

En matière d'apprentissage profond, la tâche la plus simple que l'on puisse vouloir résoudre est la classification. Il s'agit de l'association d'une étiquette (la sortie attendue) à une image (l'entrée), telle qu'une catégorie ou une classe à laquelle l'image peut être apparentée. Le *benchmark* le plus connu est « ImageNet Large-Scale Visual Recognition Challenge » (ILSVRC), basé sur le jeu de données ImageNet[11] (Deng *et al.*, 2009). Ce *challenge* propose d'identifier 1000 classes différentes correspondant à des concepts communs (animaux, personnes, véhicules, etc.), et c'est grâce à celui-ci que l'apprentissage profond a montré son potentiel et est devenu populaire pour le traitement d'images. La classification des images du jeu de données ImageNet est encore aujourd'hui une référence pour évaluer un nouvel algorithme ou une nouvelle architecture de réseau de neurones.

Une seconde tâche populaire est la segmentation, qui consiste à associer une étiquette à chaque pixel ; c'est donc une classification des pixels de l'image (Figure 2.2). Deux types de segmentation sont différenciés. La segmentation sémantique (*semantic segmentation*) permet d'identifier la catégorie de chaque pixel ; tous les pixels correspondant à une catégorie sont donc associés à la même étiquette. La segmentation d'instances (*instance segmentation*) identifie de manière unique chaque objet de l'image, même s'ils appar-

11. `https://www.image-net.org/`

FIGURE 2.2 – Exemples d'images (gauche) et leurs versions annotées (droite) pour la segmentation provenant du jeu de données MS COCO (Lin *et al.*, 2014).

tiennent à la même catégorie. Dans les deux cas, les masques de segmentation associés à chaque objet permettent de localiser précisément les objets dans les images, au pixel près. Cependant, l'apprentissage supervisé nécessitant de fournir des images annotées, la tâche de segmentation souffre souvent d'un manque de données annotées. L'annotation de chaque région de l'image est en effet longue et fastidieuse. Le*benchmark* principalement utilisé pour la segmentation est donc de taille bien plus modeste qu'ImageNet : Microsoft Common Objects in Context (MS COCO)[12] (Lin *et al.*, 2014) contient 80 classes, et un peu plus de 200 000 images annotées.

Enfin, il existe une tâche intermédiaire entre la classification et la segmentation : la détection d'objets. La détection combine l'identification et la localisation des objets dans une image, en faisant pour chacun la prédiction de sa classe ainsi que sa position grâce au rectangle englobant dans lequel il s'inscrit (Figure 2.3). Cette classe peut correspondre à une catégorie d'objets, comme les personnes, les chats ou les voitures, ou bien elle peut correspondre à une instance spécifique et unique, comme une voiture de modèle et marque donnés. Le rectangle qui entoure un objet est appelé une « boîte englobante » (*bounding box* en anglais). Dans l'approche standard, la boîte englobante a des côtés parallèles aux bords de l'image ; elle est localisée avec les coordonnées de deux sommets opposés, ou bien par les coordonnées de son centre ainsi que sa hauteur et sa largeur. Ce paradigme provient des premières méthodes de détection d'objets, qui utilisent une fenêtre glissante sur l'image pour identifier si un objet s'y trouve. Bien que le résultat d'un algorithme de

12. `https://cocodataset.org`

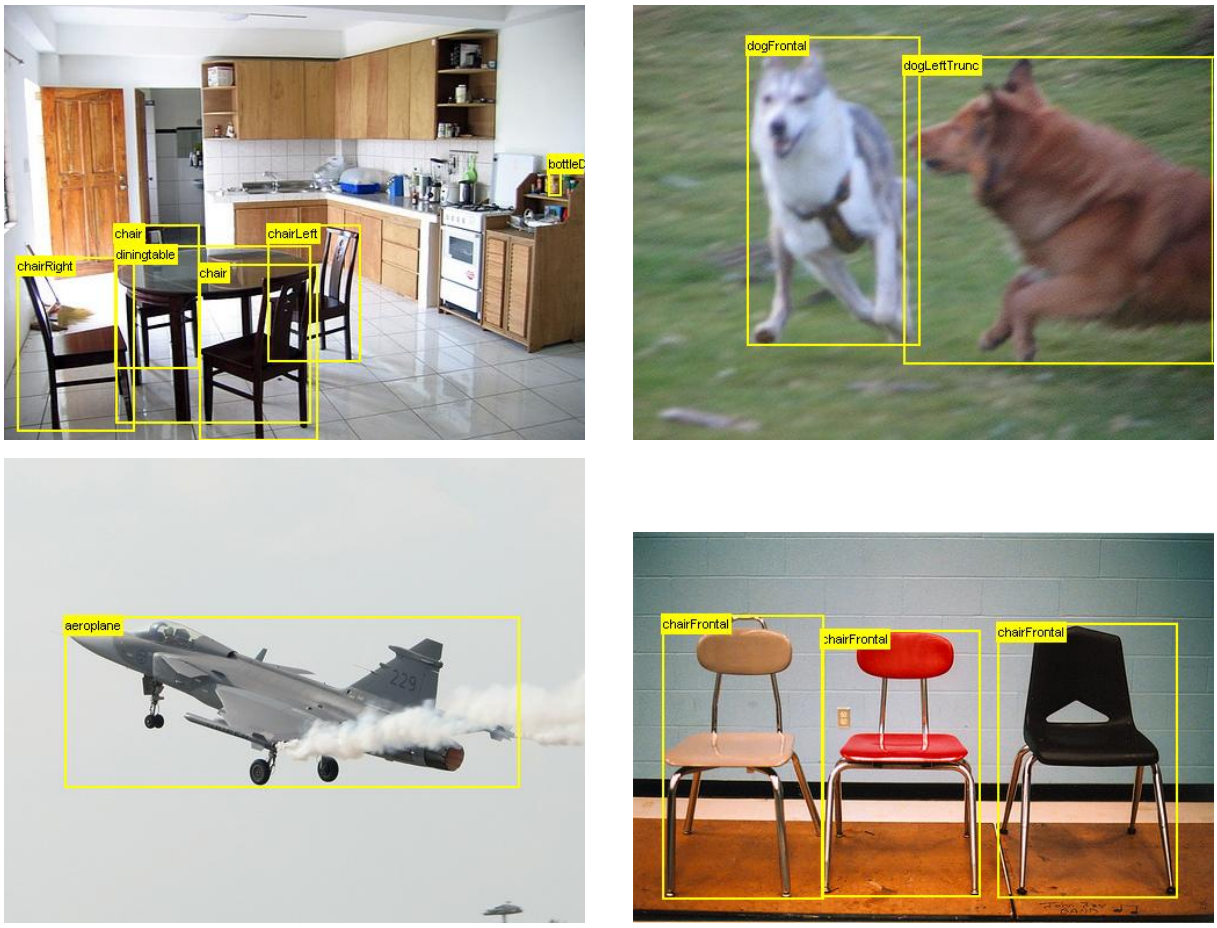


FIGURE 2.3 – Exemples d'images annotées pour la détection d'objets provenant du jeu de données Pascal VOC (Everingham *et al.*, 2010).

détection d'objets soit moins précis qu'une segmentation, les annotations sont beaucoup plus simples et rapides à obtenir. Le jeu de données Pascal VOC, créé initialement dans le cadre du Pascal Visual Object Classes *challenge* [13] (Everingham *et al.*, 2010), contient 20 classes annotées. MS COCO est aussi utilisé comme *benchmark* pour la tâche de détection d'objets avec ses 80 classes, et un ensemble d'images issues d'ImageNet a également été annoté avec des rectangles englobants pour le plus récent « ImageNet Object Localization Challenge » [14]. Une caractéristique commune de ces jeux de données est qu'ils contiennent des objets du quotidien, tels que des meubles ou des animaux. Bien que les applications industrielles n'ont en général pas besoin de détecter ces éléments-là, ces jeux de données sont très utiles pour apprendre ce qui caractérise des images naturelles. D'autres jeux de données existent pour répondre aux besoins d'autres tâches ou à des problèmes plus précis ; ils peuvent comporter différents types d'annotations et ainsi être utilisés pour résoudre différentes tâches. Par exemple, les annotations des jeux de données destinés à l'estimation de pose d'objets peuvent être utilisées pour la détection ou la segmentation d'objets ; le « Benchmark for 6D Object Pose Estimation [15] » (BOP) et ses 12 jeux de données en est un exemple. Nous revenons plus en détail sur la classification et la détection d'objets dans la section 2.3.

13. `http://host.robots.ox.ac.uk/pascal/VOC/index.html`
14. `https://www.kaggle.com/competitions/imagenet-object-localization-challenge`
15. `https://bop.felk.cvut.cz/home/`

Dans ce travail, nous nous appuyons sur les méthodes de détection d'objets avec des réseaux de neurones artificiels, afin de localiser et d'identifier des objets industriels dans des images égocentriques. Les prochains paragraphes définissent de façon plus formelle ces modèles mathématiques, ainsi que les réseaux spécifiques de la littérature sur lesquels que nous avons utilisés dans cette thèse.

2.1.2 Réseaux de neurones artificiels

2.1.2.1 Le perceptron

Un neurone artificiel est un modèle mathématique inspiré du neurone biologique, qui compose notamment le cerveau humain (Figure 2.4). Un neurone biologique accumule des signaux d'entrées via ses dendrites, et les évalue par rapport à un seuil d'activation. Si la somme des signaux est supérieure à ce seuil, le neurone est activé et envoie un signal le long de son axone ; si le seuil n'est pas atteint, aucun signal n'est propagé. Les entrées et sorties sont connectées à d'autres neurones, formant ainsi un réseau de neurones traitant l'information. Cette modélisation est formalisée par le concept du perceptron (Rosenblatt, 1958), un système nerveux hypothétique dans lequel les connexions entre les neurones se forment et s'ajustent au fur et à mesure de l'exposition à des stimulations. Ces connexions sont assimilées à des poids, qui amplifient l'information entre certains neurones, associant ainsi des groupes de neurones à des stimulations spécifiques.

FIGURE 2.4 – Illustration d'un neurone biologique[16] et d'un neurone artificiel.

Le neurone artificiel est défini de la même façon : il reçoit des signaux d'entrées qui sont multipliés par des poids représentant les connexions avec les neurones précédents, évalue leur somme avec une fonction d'activation, et envoie une sortie qui sera traitée par les neurones suivants. Ce processus est formellement défini par l'équation suivante

$$y = f(\sum_{i=1}^{n} w_i \cdot x_i + b) \tag{2.1}$$

dans laquelle y représente la sortie d'un neurone, x_i les multiples signaux d'entrée provenant des n neurones précédents, w_i les poids (ou paramètres) associés à chaque signal d'entrée (en d'autres termes, la force de la connexion avec le neurone précédent i). Le paramètre b est le biais, un poids associé au neurone, qui représente la résistance du neurone à être activé. La fonction f est une fonction d'activation, une fonction non linéaire qui définit la valeur de sortie du neurone, généralement par une saturation. Initialement,

16. https://commons.wikimedia.org/wiki/File:Neurone_biologique.JPG

cette fonction était un simple seuil faisant varier la sortie de 0 à 1 ; d'autres fonctions d'activation usuelles sont présentées dans le Tableau 2.1. Ce modèle de neurone artificiel est appelé un perceptron, d'après son modèle biologique.

TABLEAU 2.1 – Fonctions d'activation utilisées dans les réseaux de neurones artificiels.

Fonction	Equation
Seuil	$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{sinon.} \end{cases}$
Sigmoïde/Logistique	$f(x) = \frac{1}{1+\mathrm{e}^{-x}}$
Tangente hyperbolique (*tanh*)	$f(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$
ReLU (*Rectified Linear Unit*)	$f(x) = max(0, x)$
Leaky ReLU	$f(x) = max(\alpha x, x) \quad \text{avec} \alpha \in]0, 1]$
Softmax	$f(x)_i = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}}$

2.1.2.2 Le perceptron multicouche

Le perceptron peut être utilisé seul afin de modéliser une relation simple entre des données d'entrée et des sorties. Cependant, il ne s'ajustera pas à une relation plus complexe. Le modèle du perceptron multicouche a donc été défini pour cela (dénoté MLP, pour *Multi-Layer Perceptron* en anglais), toujours inspiré des neurones biologiques (Rosenblatt, 1958). Le MLP est une organisation de plusieurs perceptrons en une série de couches, à travers lesquelles l'information d'entrée est propagée jusqu'à la sortie de la dernière couche. L'organisation spécifique des couches, comme le nombre de neurones, définit l'architecture du réseau ; toute modification dans cette organisation correspond à une architecture différente, ou à une variation d'une architecture de base. On considère généralement trois types de couches : la couche d'entrée, qui correspond aux valeurs du signal d'entrée ; la couche de sortie, qui donne la prédiction du réseau ; la, ou les, couches intermédiaires, chacune composée d'un ou plusieurs neurones. Le réseau de neurones le plus simple correspond à un MLP à une seule couche intermédiaire. Plus on insère de couches successives entre l'entrée et la sortie, et plus le réseau devient « profond », d'où le terme d'apprentissage profond. Un réseau MLP à deux couches intermédiaires est représenté sur la Figure 2.5. Ce type de réseau de neurones peut être utilisé pour la classification d'un signal : les entrées x_i représentent les valeurs du signal sous forme de vecteur, et le vecteur de sortie représente la probabilité que l'entrée x corresponde à chaque catégorie y_i. Ces probabilités, dont la somme vaut un, sont obtenues en remplaçant la fonction d'activation de la dernière couche par une couche *softmax*. En considérant x l'entrée de la dernière couche, et K le nombre total de classes (et donc de neurones de sortie), chaque sortie est

calculée par l'équation *softmax* suivante :

$$y_i = p(y = i|x) = \frac{\mathrm{e}^{(w_i^T x + b_i)}}{\sum_{k=1}^{K} \mathrm{e}^{(w_k^T x + b_k)}} \tag{2.2}$$

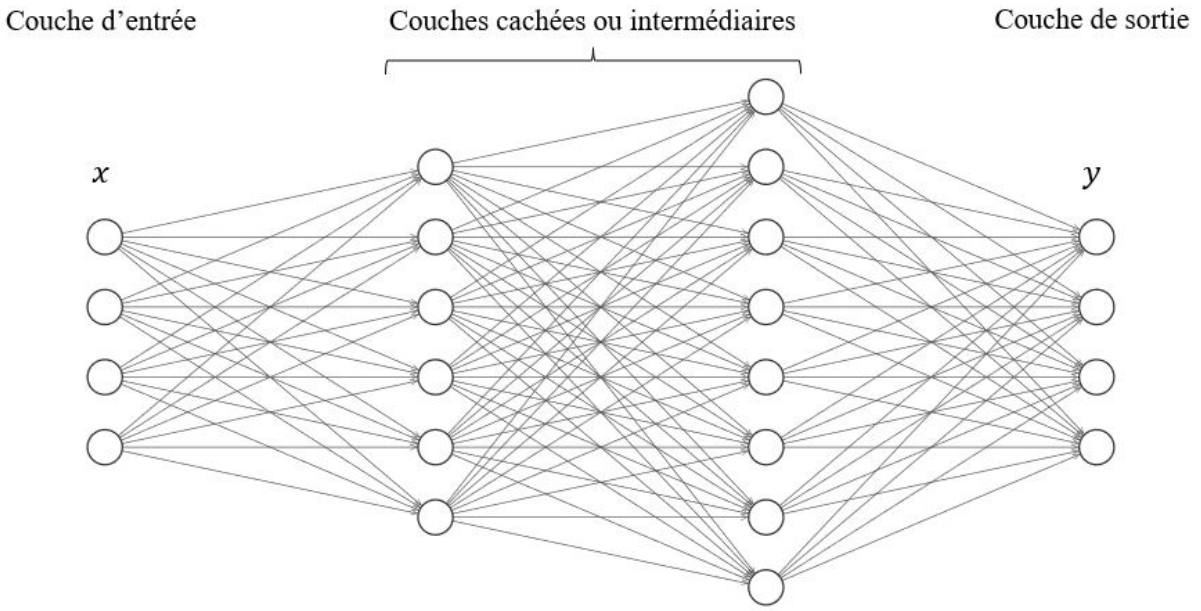


FIGURE 2.5 – Représentation d'un réseau de neurones artificiel de type perceptron multicouche (MLP). Ce réseau a 4 entrées, 2 couches cachées de 6 et 8 neurones respectivement, et une couche de sortie de 4 neurones (pour une classification à 4 classes par exemple).

Dans ces réseaux de neurones, chaque neurone d'une couche est lié par un poids à chaque neurone de la couche précédente et de la couche suivante. Ces couches portent donc le nom de couches complètement connectées, et sont notées couches « FC » pour *fully connected* en anglais. Un réseau composé de couches FC est appelé un réseau complètement connecté. Une couche FC qui comporte n entrées et m neurones, chacun associé à un biais, contient donc $(n+1) \cdot m$ paramètres. Lorsque l'entrée du réseau comporte un grand nombre d'éléments, le nombre de paramètres augmente fortement, et avec lui le nombre d'opérations nécessaires jusqu'à la sortie. Cette raison rend les réseaux complètement connectés peu adaptés aux types de données comme les images, pour lesquelles un pixel correspond à une entrée. Si une petite image de 32×32 pixels contient « seulement » 1024 poids associés à la première couche, ce nombre passe à 921 600 paramètres pour une image en haute définition de 1280×720 pixels (ce qui est encore en-dessous de la définition des *smartphones* actuels). Ces réseaux ne peuvent donc pas être utilisés pour des images sans que leur taille n'explose. Cela a poussé les chercheurs à proposer une autre approche pour traiter les images, plus raisonnable en nombre de paramètres : les réseaux de neurones à convolution. En considérant les images comme des matrices, et non comme des vecteurs, ils prennent également mieux en compte les relations spatiales des éléments représentés.

2.1.2.3 Réseaux de neurones à convolutions

De façon intuitive, une convolution est une somme pondérée des valeurs dans le voisinage d'un pixel considéré, qui correspond à une région d'une image par exemple. Mathé-

matiquement, cela correspond à la combinaison linéaire des éléments d'un noyau avec les éléments d'une matrice. C'est la notion à la base de nombreuses méthodes de traitement d'images : en choisissant soigneusement les poids du noyau et en l'appliquant à chaque pixel d'une image, la convolution permet d'éliminer différents types de bruit, de rendre une image plus floue, ou au contraire d'en faire ressortir les contours.

Dans un réseau de neurones à convolutions (noté CNN pour *Convolutional Neural Network* en anglais), les paramètres ne sont plus organisés de façon complètement connectée, mais sous forme d'un noyau de convolution, appliqué comme une fenêtre glissante sur toute l'image. Ainsi, la convolution du noyau avec chaque région de l'image donne un nombre scalaire, et l'ensemble de ces scalaires donne une nouvelle image appelée « carte de caractéristiques » (ou *feature map* en anglais). Ces noyaux sont appelés des filtres de convolution ; leurs poids sont appris, et l'on définit autant de noyaux que l'on souhaite de canaux dans la carte de caractéristiques en sortie. Ensuite, on applique une nouvelle série de filtres sur cette carte de caractéristiques, et ainsi de suite jusqu'à obtenir la sortie souhaitée. Les filtres de convolution forment une couche de convolution, et une séquence de couches de convolution forme un CNN, comme illustré sur la Figure 2.6. Celle-ci montre une architecture classique, composée d'une couche de convolution (dans laquelle est inclue la fonction d'activation), d'une couche de *pooling*, d'une seconde couche de convolution, d'une mise à plat des couches de caractéristiques pour créer un vecteur, puis d'une couche complètement connectée. Les couches de *pooling* et l'opération de mise à plat sont décrites dans la suite de cette section.

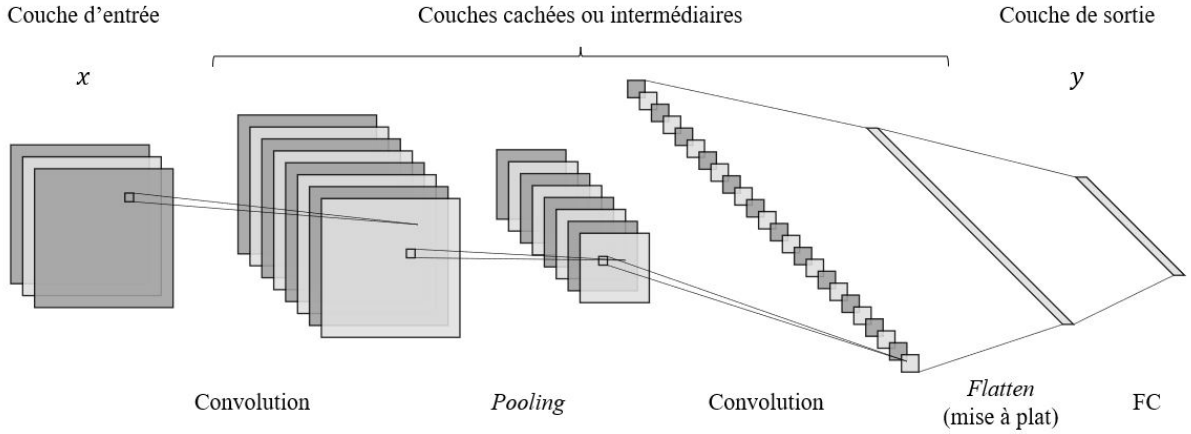


FIGURE 2.6 – Représentation d'un réseau de neurones à convolution. L'entrée est une image à 3 canaux ; le réseau est composé d'une première couche de convolution (avec fonction d'activation), d'une couche de *pooling*, d'une seconde couche de convolution, puis d'une couche *flatten* (qui transforme les cartes de caractéristiques en vecteur), et enfin d'une couche complètement connectée (FC) afin de prédire la sortie.

Chaque couche est définie par plusieurs paramètres : la taille des noyaux qui la composent, le pas (ou *stride* en anglais), et le *padding*. Un noyau de taille D_K contient en général $D_K \times D_K$ paramètres, D_K étant défini comme un nombre impair afin de contenir un pixel central. Dans certaines applications, le noyau peut être défini avec une hauteur et une largeur différentes, par exemple quand le type d'imagerie n'est pas régulier (Douarre *et al.*, 2021). Ce noyau de convolution est déplacé sur toute l'image en se décalant verticalement et horizontalement d'un nombre de pixels égal au pas. Un pas supérieur à 1 permet

de sous-échantillonner l'image, et ainsi de contrôler sa taille de sortie. Enfin, un noyau de taille supérieure à 1 ne peut être appliqué sur les pixels du bord de l'image, car il dépasse de celle-ci. La solution proposée à cela est le *padding*, qui correspond à l'extension des bords de l'image de quelques pixels. Définir le *padding* correspond à définir le nombre de pixels, ainsi que la méthode d'extension choisie : une copie des pixels du bord de l'image, une symétrie, etc.

D'autres types de couches peuvent être insérées dans un réseau de neurones à convolutions. Tout comme les réseaux complètement connectés, chaque élément constituant une couche de caractéristiques passe au travers d'une fonction d'activation, toujours afin de limiter leur amplitude. Ensuite, les réseaux convolutionnels contiennent généralement des couches de *pooling*, bien que leur utilisation commence à être remise en question (Yu et Koltun, 2015; Ruderman *et al.*, 2018). L'opération de *pooling* correspond à la réduction de la dimension spatiale d'une couche de caractéristiques, ceci afin de réduire la place occupée par le réseau. En effet, à mesure que l'on propage une image dans un réseau à convolution, la profondeur des couches de convolution a généralement tendance à augmenter afin d'extraire un grand nombre de caractéristiques différentes. La réduction de la définition spatiale permet d'éviter l'explosion du nombre de paramètres du réseau avec l'augmentation de sa profondeur, ainsi que l'extraction de caractéristiques correspondant à différentes échelles dans l'image. Intuitivement, les premières couches du réseau identifient des caractéristiques simples comme les bords et les hautes fréquences, et les dernières couches correspondent à des informations de plus haut niveau sémantique, moins sensibles aux détails de l'image. Une couche de *pooling* correspond à l'application d'une opération sur un voisinage de pixels, avec une fenêtre glissante sans superposition, afin de produire un scalaire pour chaque région. La sortie sera donc un sous-échantillonnage de l'entrée. Les opérations couramment utilisées sont les suivantes :

- *average-pooling* (*pooling* par moyenne) : sélection de la valeur moyenne sur un voisinage.
- *min-pooling* : sélection de la valeur minimum sur un voisinage.
- *max-pooling* : sélection de la valeur maximum sur un voisinage.

La pratique courante les dernières années était d'utiliser le *max-pooling*, car il permet de ne garder que les éléments les plus saillants d'une couche de caractéristiques. Il est cependant possible de s'en passer en utilisant différents types de convolution, qui permettent de sous-échantillonner une carte de caractéristiques sans avoir recours au *pooling* (Springenberg *et al.*, 2014).

Un réseau dit « complètement convolutif » est composé uniquement de couches de convolution, d'activation et de *pooling*. L'avantage est qu'il peut prendre en entrée des images de tailles variables, puisque la taille des noyaux ne dépend pas de la taille de l'entrée. Dans la pratique, la sortie du réseau nécessite souvent une ou plusieurs couches complètement connectées, afin de produire la distribution de probabilité des classes par exemple. Dans l'exemple de la Figure 2.6, une couche *flatten* est introduite : c'est une opération de transformation de couches de caractéristiques en un vecteur, afin de permettre l'application de la couche complètement connectée par la suite. S'il ne s'agit pas techniquement d'une couche du réseau, mais plutôt d'une opération « pratique », on l'appelle toutefois « couche » puisque les caractéristiques passent à travers, et leur forme en est modifiée.

2.1.2.4 Différents types de convolutions

Différents types de convolutions ont été proposés dans la littérature, que nous détaillons ci-dessous :

- Convolution classique (Figure 2.7a) : c'est le filtre de convolution le plus général. Il est composé de N noyaux de tailles $D_K \times D_K \times M$, chacun étant appliqué aux M canaux de l'image d'entrée afin de produire une carte de caractéristiques de profondeur 1. Les N cartes de profondeur 1 sont ensuite concaténées, produisant une carte de caractéristiques de profondeur N.
- Convolution en profondeur (*depthwise convolution*, Figure 2.7b) : chaque noyau de convolution est de profondeur 1 (au lieu du nombre de canaux d'entrée), et est appliqué à un canal différent de l'image d'entrée, produisant autant de canaux en sortie, qui sont concaténés. En d'autres termes, un noyau différent est appliqué à chaque canal d'entrée, et la profondeur de la sortie est égale à celle de l'entrée.
- Convolution ponctuelle ou 1×1 (*pointwise convolution*, Figure 2.7c) : le filtre de convolution a une taille D_K égale à 1 pixel, ce qui correspond à une combinaison linéaire des canaux de l'image d'entrée.
- Convolution séparable en profondeur (*depthwise separable convolution*, Figure 2.7d) : il s'agit de la décomposition d'une convolution standard en une convolution en profondeur suivie d'une convolution ponctuelle. Cela a pour effet de réduire très fortement le nombre d'opérations et la taille du modèle, pour une perte de performance négligeable (Howard *et al.*, 2017). Une convolution standard nécessite un nombre d'opérations égal à $D_K \times D_K \times M \times N \times D_F \times D_F$, avec D_F la hauteur et largeur de la couche d'entrée F (en supposant une image carrée), M le nombre de canaux de la couche d'entrée, et N celui de la couche de sortie. Une convolution séparable en profondeur a un coût de calcul de $D_K \times D_K \times M \times D_F \times D_F$ pour la première étape (indépendante du nombre de canaux de sortie N) et de $M \times N \times D_F \times D_F$ pour la seconde étape (indépendante de la taille du noyau D_K). Le coût total est donc de $M \times D_F \times D_F \times (D_K \times D_K + N)$.

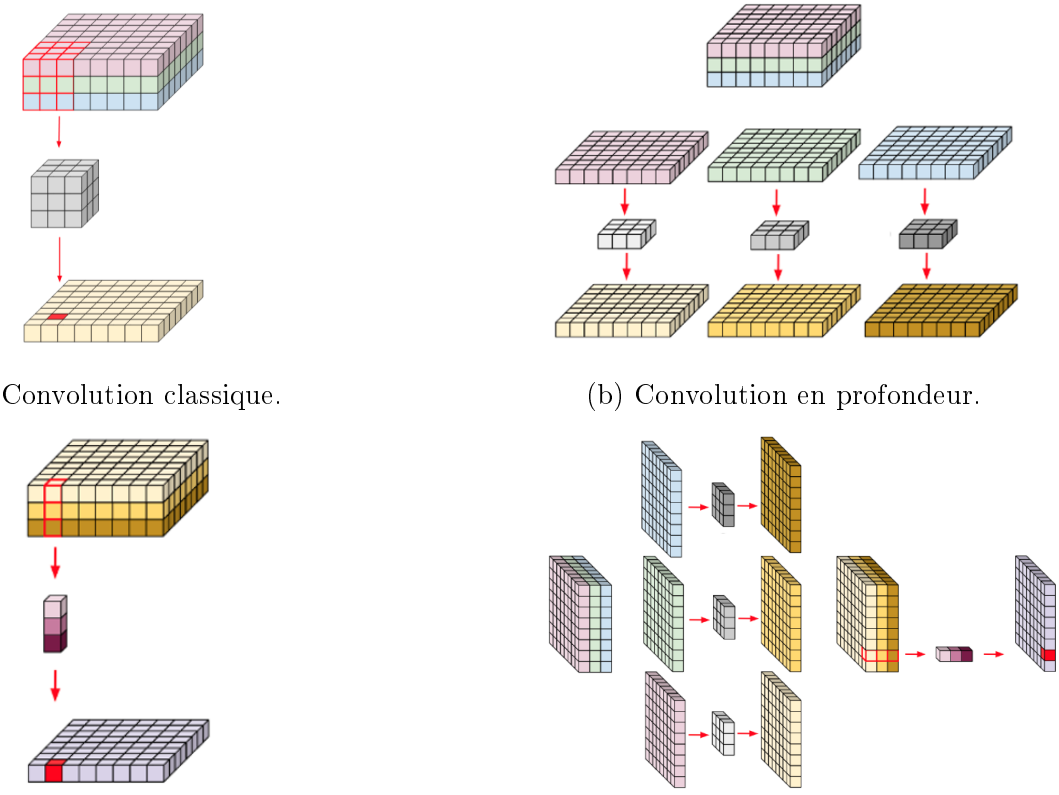
Ces différentes couches de convolution sont combinées dans les architectures de réseaux de neurones, afin de maximiser la capacité d'apprentissage de celui-ci tout en minimisant le nombre de poids, d'opérations, le temps d'entraînement et le temps d'inférence.

2.1.3 Extraction de caractéristiques

Bien qu'en apprentissage profond, l'étape d'extraction des caractéristiques ne soit pas explicite ni faite à l'aide d'algorithmes *ad hoc*, elle est tout de même présente à l'intérieur du réseau de neurones. En effet, un réseau de neurones à convolutions peut être décomposé en deux parties : un extracteur de caractéristiques (aussi appelé *backbone*), et un ensemble de couches dédiées à une tâche en particulier. Intuitivement, le réseau extrait d'abord des caractéristiques des images en entrée, avant de les assembler pour résoudre le problème abordé.

Ainsi, un certain nombre d'architectures ont été proposées dans la littérature afin d'extraire des caractéristiques de façon optimale. Ces architectures sont généralement définies comme des classifieurs, des réseaux pour la classification d'images. En réalité, les dernières

17. `https://chingisoinar.medium.com/important-cnn-concepts-you-should-know-in-2022-ec914942fd82`



(a) Convolution classique.

(b) Convolution en profondeur.

(c) Convolution ponctuelle.

(d) Convolution séparable en profondeur.

FIGURE 2.7 – Illustration des différents types de convolutions [17].

couches seulement sont dédiées à la classification, et le reste du réseau peut être réutilisé pour d'autres tâches. Cette décomposition permet d'introduire la notion d'apprentissage par transfert : un réseau est entraîné avec un jeu de données pour résoudre une tâche, puis il est réutilisé dans un autre contexte (autre tâche et/ou jeu de données).

On définit la phase de pré-entraînement comme l'entraînement du réseau initial sur un jeu de données très grand et varié, avant de le réutiliser pour la tâche ou le jeu de données qui nous intéresse. C'est la classification du *benchmark* ImageNet qui est utilisée comme tâche de pré-entraînement pour une majorité d'autres tâches liées à l'analyse d'images (Huh *et al.*, 2016; Razavian *et al.*, 2014). Une partie des couches est ensuite extraite, formant le réseau pré-entraîné, pour intégrer une autre architecture, pour la détection ou la segmentation par exemple, mais également pour la classification avec un nombre différent de classes. Ainsi, l'entraînement pour la tâche principale ne démarre pas de poids initialisés aléatoirement, mais de poids pré-entraînés. Ce n'est pas nécessaire pour toutes les applications, mais très utile dans de nombreux contextes. Les avantages sont multiples : moins de données sont nécessaires pendant l'entraînement, et celui-ci est également beaucoup plus rapide. En effet, la grande diversité dans les images d'ImageNet permet au *backbone* d'apprendre un grand nombre de schémas, formes et textures, à travers les filtres pré-entraînés et réutilisés.

Enfin, la notion de *fine-tuning*, ou raffinement des poids, fait référence à l'entraînement de tout ou partie d'un réseau déjà entraîné, en général avec des paramètres différents, afin de spécialiser le réseau sur ces données. Le pré-entraînement peut être vu comme une modélisation très générale des données, et le *fine-tuning* comme une spécialisation sur un jeu de données particulier.

La visualisation des noyaux d'un réseau à convolutions après l'entraînement donne une idée des caractéristiques apprises par ce réseau. La Figure 2.8 montre des caractéristiques apprises par le réseau AlexNet entraîné pour la classification d'ImageNet (Krizhevsky *et al.*, 2012). On y voit des schémas très différents, qui vont permettre au réseau d'identifier des objets d'apparences différentes.

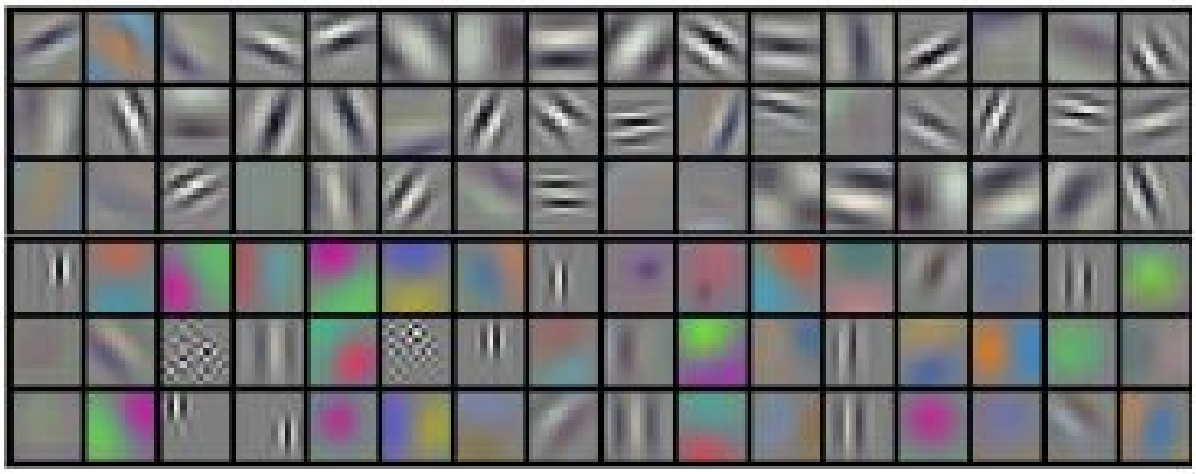


FIGURE 2.8 – Visualisation des 96 noyaux de tailles $11 \times 11 \times 3$ de la première couche du réseau AlexNet entraîné avec ImageNet (Krizhevsky *et al.*, 2012).

2.1.4 Les auto-encodeurs

Bien que nous nous intéressons aux réseaux de détection d'objets entraînés de façon supervisée, un autre type de réseaux de neurones mérite d'être introduit pour son importance dans de très nombreuses applications : l'auto-encodeur (Kramer, 1991). Un auto-encodeur est un assemblage de deux sous-réseaux, nommés l'encodeur et le décodeur. Son but est, comme son nom l'indique, d'encoder des données d'entrée dans une représentation compressée, telle qu'un vecteur de caractéristiques, appelé « code latent » ou « représentation latente ». Le décodeur prend en entrée ce code latent, et a pour mission de reconstruire l'entrée à partir de cette représentation compressée. Initialement dédiés à la réduction de dimensionnalité et entraînés couche par couche, les auto-encodeurs (notés AE) sont aujourd'hui entraînés de bout en bouts (Krizhevsky et Hinton, 2011; Masci *et al.*, 2011). Ainsi, de nombreuses variations de l'auto-encodeur simple existent, notamment celle du *denoising autoencoder* consistant à perturber l'image d'entrée (par exemple avec du bruit ou en cachant une partie de l'image), et en poussant le décodeur à reconstruire l'image d'origine, non perturbé (Vincent *et al.*, 2008, 2010). Cette tâche simple est très puissante puisqu'elle permet à l'encodeur d'apprendre des caractéristiques robustes sans nécessiter d'annotations. Les auto-encodeurs ont ensuite été utilisés pour pré-entraîner des réseaux de neurones, et étendu à de nombreuses applications comme la génération de données avec les GANs (*Generative Adversarial Networks*) par exemple (Makhzani *et al.*, 2015).

2.2 L'optimisation d'un réseau de neurones

Un modèle d'apprentissage profond doit être optimisé pour une tâche sur un jeu de données avant d'être déployé. L'optimisation d'un réseau correspond à la modification de ses paramètres jusqu'à ce que les sorties soient satisfaisantes, le critère de satisfaction dépendant du contexte. Dans cette partie, nous détaillons les différentes étapes d'apprentissage des poids et d'évaluation du réseau.

2.2.1 La phase d'entraînement

Les paramètres associés à un réseau de neurones artificiels sont initialement inconnus. En effet, on ne peut pas savoir à l'avance les poids qui permettront à des données d'entrée de produire les sorties attendues. C'est pourquoi une phase d'apprentissage est nécessaire, aussi appelée entraînement ou optimisation. Elle est composée des quatre étapes suivantes, qui sont détaillées par la suite :

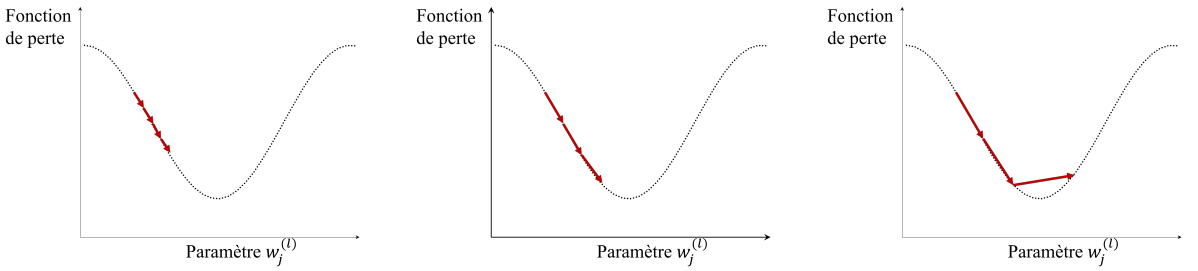
- Propagation avant : c'est le passage des données d'entrée jusqu'à la sortie du réseau, afin d'obtenir une prédiction.
- Calcul de l'erreur : la distance entre la prédiction et la sortie attendue (pour de l'apprentissage supervisé) est calculée par une fonction, appelée fonction de coût ou de perte, que l'on cherche à minimiser.
- Descente de gradient : calcul des gradients de chaque paramètre par rétro-propagation afin de minimiser l'erreur.
- Mise à jour des paramètres à partir des gradients calculés.

Les paramètres initiaux sont choisis de façon aléatoire, en suivant une distribution donnée (Glorot et Bengio, 2010; He *et al.*, 2015). Ces paramètres correspondent aux poids $w_j^{(l)}$ et aux biais $b^{(l)}$, avec (l) correspondant au numéro de la couche considérée dans le réseau, et j l'indice du poids dans cette couche. Chaque image du jeu de données est propagée dans le réseau de l'entrée jusqu'à la sortie, produisant une prédiction $\hat{y}$. Cette prédiction est comparée à la sortie attendue y (dans le cas supervisé) via la fonction de perte $\mathscr{L}(\hat{y}, y)$, qui symbolise en général une distance, dont la définition varie selon le contexte. Cette distance est minimisée par mise à jour itérative des poids par descente de gradient, un algorithme d'optimisation différentiable. Pour chaque paramètre à optimiser, cet algorithme calcule le gradient qui minimise la fonction de coût, et met à jour la valeur du paramètre comme indiqué par les équations 2.3 (pour les poids) et 2.4 (pour les biais), dans lesquelles λ correspond au « taux d'apprentissage ». L'algorithme de rétro-propagation du gradient, formellement défini par Rumelhart *et al.* (1986), a été proposé afin de calculer ces gradients en appliquant le théorème de dérivation des fonctions composées. Cette méthode exprime la prédiction $\hat{y}$ en fonction des poids de la couche de sortie, puis ceux-ci en fonction des poids de la couche précédente, et ainsi de suite. Les gradients sont donc calculés couche par couche, de la sortie du réseau vers l'entrée d'où le nom de « propagation arrière », ou rétro-propagation.

$$w_j^{(l)} = w_j^{(l)} - \lambda \frac{\partial \mathscr{L}}{\partial w_j^{(l)}} \tag{2.3}$$

$$b^{(l)} = b^{(l)} - \lambda \frac{\partial \mathscr{L}}{\partial b^{(l)}} \tag{2.4}$$

Intuitivement, λ peut être vu comme la taille du pas pris dans la direction indiquée par la dérivée partielle. Ce pas doit être correctement ajusté afin de rejoindre un minimum local de la fonction de perte suffisamment rapidement et précisément (Figure 2.9). Plusieurs analyses et discussions approfondies autour des concepts de minimum local et global pour l'apprentissage profond sont proposées par Choromanska *et al.* (2015) et Kawaguchi (2016). Des algorithmes de mise à jour des poids plus complexes ont également été proposés pour mieux contrôler l'évolution des paramètres et atteindre plus rapidement le minimum local le plus proche (Duchi *et al.*, 2011; Kingma et Ba, 2014). En particulier, le *momentum* (Qian, 1999) est un terme souvent utilisé ; il prend en compte le gradient des étapes précédentes, et pas seulement celui de l'étape actuelle, afin de mieux déterminer la direction que doit prendre le gradient.



(a) λ trop petit : nécessite de nombreuses étapes pour atteindre le minimum.

(b) λ de bonne taille : atteint le minimum rapidement.

(c) λ trop grand : n'atteint pas le minimum.

FIGURE 2.9 – Illustration de la recherche du minimum de la fonction de perte avec différents taux d'apprentissage λ.

La méthode de descente de gradient généralement utilisée pour l'apprentissage profond est qualifiée de « stochastique » (*Stochastic Gradient Descent* en anglais, notée SGD) car elle n'est pas appliquée à tout le jeu de données à chaque étape, mais à une petite quantité de données seulement. En effet, le calcul des gradients demande de stocker les valeurs intermédiaires de chaque couche lors de la propagation avant, ce qui nécessite beaucoup de place en mémoire, en particulier pour les réseaux à convolutions de très grande taille. Dans la pratique, les images passent à travers le réseau sous forme de petits lots appelés *mini-batches* ou *batches*, et le réseau est optimisé avec une approximation locale du gradient. Une itération d'entraînement correspond à l'optimisation des poids du réseau avec un *batch*, dont les données sont choisies au hasard parmi le jeu d'entraînement ou en suivant un ordre défini (on parle de *curriculum learning* (Bengio *et al.*, 2009)). Lorsque toutes les images ont été fournies au réseau de neurones une fois, cela correspond à un *epoch*. La durée de l'entraînement est définie en nombre d'*epochs* ou d'itérations, suivant la convention adoptée. L'entraînement est interrompu lorsque la performance ne s'améliore plus, mesurée avec la métrique de son choix sur l'ensemble de validation (défini dans la prochaine section), ou bien lorsque la fonction de coût est stabilisée et ne diminue plus. Lors de cette phase, les paramètres de la méthode d'optimisation, le taux d'apprentissage λ, le nombre d'*epochs* ou encore la taille du *batch* sont appelés « hyperparamètres », et influent fortement sur l'optimisation du réseau. Ils sont choisis lors de la phase de validation.

2.2.2 La phase de validation et d'évaluation

Pendant l'entraînement, l'apprentissage du réseau doit être suivi afin de s'assurer que les poids évoluent dans la bonne direction. La fonction de perte est un premier indicateur à observer : elle doit diminuer au fur et à mesure de l'entraînement. En général, elle subit une chute importante pendant les premiers *epochs*, puis se stabilise. Une forme différente de cela indique un problème, comme un taux d'apprentissage pas adapté par exemple.

Ensuite, la performance du modèle doit être évaluée avec une métrique spécifique, encore une fois dépendante du contexte. Cette métrique n'est pas calculée directement sur les données d'entraînement, mais sur un jeu de données différent appelé « ensemble de validation ». Ces données de validation correspondent à des données qui n'ont pas été fournies au réseau pendant l'entraînement. En effet, la grande capacité d'apprentissage d'un réseau de neurones peut le pousser au sur-apprentissage (ou *overfitting*), une situation dans laquelle le modèle « apprend par cœur » les données d'entraînement. Alors que le réseau est sensé apprendre la distribution statistique du jeu de données, il doit également être capable de généraliser à des données de même distribution mais qu'il n'a jamais « vu ». En cas de sur-apprentissage, la distribution statistique apprise colle trop précisément aux données « connues », et ne généralise pas du tout aux nouvelles données. L'exemple de la Figure 2.10 illustre cette situation dans le cas de la régression d'un polynôme. Une courbe qui s'ajuste trop précisément aux données d'entraînement correspond à du sur-apprentissage, et ne pourra pas prédire correctement la sortie pour un nouvel exemple. La méthode de validation croisée (ou *k-fold cross validation*) consiste à définir plusieurs sous-ensembles de données qui sont utilisés à la fois pour l'entraînement et la validation, de façon alternée, afin d'obtenir une performance moyenne avec différents sous-ensembles des données pour entraîner et tester le modèle.

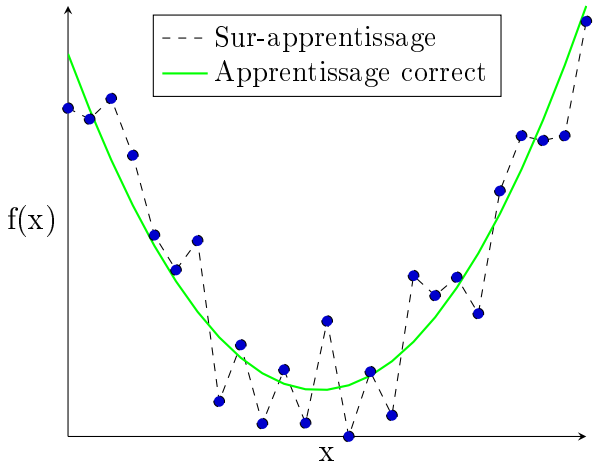


FIGURE 2.10 – Illustration d'un modèle ajusté aux données (en vert) et d'une situation de sur-apprentissage (courbe en pointillés noirs). Les données d'entraînement correspondent aux cercles pleins.

Plusieurs techniques permettent d'éviter ou de limiter le sur-apprentissage, ce que l'on appelle la « régularisation » du réseau :

- Régularisation L2 (ou *weight decay*) (Krogh et Hertz, 1991; Bartlett, 1996) : un terme de pénalité correspondant à la somme des carrés des valeurs des paramètres est ajouté à la fonction de perte, afin d'empêcher les poids de prendre une valeur

trop élevée. Son importance est modulée par un coefficient, considéré comme un hyperparamètre.

- Pré-entraînement non-supervisé (Erhan *et al.*, 2010) : avant l'utilisation du jeu de données ImageNet pour le pré-entraînement des réseaux de neurones, ils étaient initialisés à l'aide d'une tâche non supervisée. Les expériences de Erhan *et al.* (2010) ont montré que les poids étaient ainsi initialisés de façon à atteindre un meilleur minimum local par la suite, menant à une meilleure capacité de généralisation.
- *Dropout* (Srivastava *et al.*, 2014) : cela consiste à désactiver des neurones (et leurs connexions) de façon aléatoire dans une couche complètement connectée pendant l'entraînement. Cela pousse les neurones à s'adapter à des configurations différentes.
- *Batch Normalization* (Ioffe et Szegedy, 2015) : une couche *BatchNorm* normalise la distribution des entrées de chaque couche. A chaque mise à jour des paramètres d'une couche, la distribution qui en sort est différente, ce qui rend l'entraînement plus difficile pour la couche suivante, qui doit sans cesse s'adapter. Normaliser les entrées des couches permet de réduire ces variations, accélérer et faciliter l'entraînement.

Au-delà de détecter une situation de sur-apprentissage, l'ensemble de validation sert aussi à choisir les hyperparamètres les plus adaptés pour un réseau et un jeu de données spécifiques. Dans la pratique, plusieurs expériences sont réalisées en faisant varier l'un de ces paramètres à la fois, soit durant un entraînement complet, soit pendant un petit nombre d'*epochs* uniquement. Plusieurs méthodes existent pour tenter de trouver les meilleurs paramètres : la recherche par grille (*Grid Search*) teste de façon exhaustive les combinaisons de paramètres spécifiées ; la recherche aléatoire (*Random Search*) teste des valeurs aléatoires pour chaque paramètre (dans un intervalle ou ensemble de valeurs pré-défini). En général, la recherche aléatoire permet de réduire l'espace des paramètres plus rapidement, avant d'effectuer une recherche par grille pour seulement quelques valeurs dans un intervalle plus réduit. Les paramètres peuvent également être modifiés pendant l'entraînement, lorsque la performance stagne à l'approche d'un minimum (Smith *et al.*, 2017). Il est aussi possible de faire varier l'architecture ou comparer des extracteurs de caractéristiques différents, toujours en utilisant l'ensemble de validation pour maximiser la performance.

Malgré la séparation entre les images d'entraînement et de validation, la performance mesurée n'est toutefois pas encore représentative de la performance que l'on peut espérer lorsque le modèle sera déployé dans le monde réel. En effet, les hyperparamètres sont ajustés sur l'ensemble de validation, qui n'est qu'un échantillon de la distribution que l'on souhaite modéliser. Il est donc possible que la performance sur d'autres exemples ne soit pas la même. Afin d'avoir une mesure la plus réaliste possible de la performance « réelle » du réseau, on définit donc un troisième jeu de données, appelé « jeu de test », sur lequel le réseau de neurones est testé une fois que tous les paramètres sont validés. Cette performance est celle qui sert de référence lorsque l'on compare le réseau à d'autres travaux de la littérature par exemple. Dans les *challenges* et *benchmarks*, il est même courant de ne pas fournir aux participants les annotations correspondant aux données de test, voire parfois de ne pas fournir du tout les données de test, afin d'éviter leur utilisation pendant l'optimisation du réseau.

2.2.3 Mesures de performance

De nombreuses métriques ont été définies afin d'évaluer les prédictions de différents algorithmes. Nous définissons ici les mesures principales utilisées dans le cadre de la clas-

sification et de la détection d'objets, souvent appelées « scores ».

Tout d'abord, il est important de définir les métriques de base : la précision et le rappel. Elles s'appuient sur la catégorisation des prédictions et des sorties attendues en quatre catégories (illustrées sur la Figure 2.11) :

- Vrai Positif (TP pour *True Positive* en anglais) : la classe prédite correspond à celle de la vérité-terrain.
- Faux Positif (FP, *False Positive*) : la classe prédite ne correspond pas à celle de la vérité-terrain.
- Faux Négatif (FN, *False Negative*) : un élément de la vérité terrain n'a pas été prédit, il n'a pas de prédiction correcte associée.
- Vrai Négatif (TN, *True Negative*) : un élément n'a pas été détecté à juste titre, ou correspond à la classe négative.

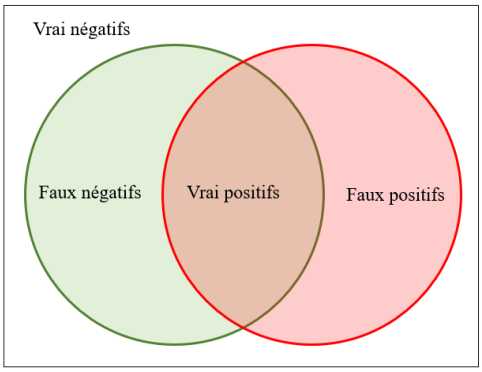


FIGURE 2.11 – Illustration de la répartition des éléments de la vérité-terrain (cercle vert) et des prédiction (cercle rouge) dans les différentes catégories.

Prenons comme exemple le cas de la détection d'anomalies dans une image. Si une anomalie est présente (d'après la vérité terrain) et a été détectée (prédiction), c'est un vrai positif. Si la prédiction indique qu'il y a une anomalie alors que ce n'est pas le cas, c'est un faux positif. Si une anomalie est présente mais non détectée, c'est un faux négatif. Enfin, s'il n'y a pas d'anomalie et que cela correspond bien à la prédiction, alors c'est un vrai négatif. Cette catégorisation des échantillons permet de définir la précision (P) et le rappel (R) comme suit :

$$P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN} \tag{2.5}$$

La précision correspond au taux de prédictions correctes parmi toutes les prédictions, alors que le rappel indique les prédictions correctes par rapport à la vérité terrain. Pour reprendre l'exemple précédent, la précision indique la proportion de vraies anomalies parmi toutes les prédictions d'anomalies, et le rappel correspond aux anomalies effectivement détectées parmi toutes les anomalies. Ces deux métriques prennent des valeurs entre zéro et un : une précision de un indique que toutes les prédictions faites sont correctes (mais il peut en manquer), et un rappel de un indique que toutes les prédictions attendues ont été données (mais il peut y avoir des prédictions supplémentaires incorrectes). Il y a donc un compromis à trouver, qui dépend généralement de l'application.

Dans un contexte de classification, la synthèse entre ces deux mesures peut être obtenue avec le score F1, ou *F1-score*. Il est défini comme la moyenne harmonique de la

précision et du rappel, et peut s'exprimer avec le nombre de TP, FN et FP :

$$F1\text{-}score = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{TP}{TP + \frac{1}{2}(FN + FP)} \tag{2.6}$$

Pour la détection d'objets, l'identification des prédictions comme TP, FP et FN dépend de la localisation des prédictions par rapport aux boîtes englobantes de la vérité-terrain, ainsi que des catégories qui leur sont associées. En effet, une boîte englobante bien localisée mais avec la mauvaise étiquette ne peut pas être considérées comme correcte, et la notion même de « bonne localisation » peut être débattue. De plus, un score de confiance est associé à chaque prédiction, qui permet de hiérarchiser de multiples prédictions sur une même vérité-terrain. Ainsi, différentes métriques ont été proposées pour différents *challenges* et jeux de données. Récemment, elles ont été analysées et unifiées par Padilla *et al.* (2020) afin de faciliter la comparaison des algorithmes de détection d'objets. Une implémentation de ces métriques est disponible en ligne [18]. Afin d'évaluer les prédictions d'un détecteur d'objets, on définit tout d'abord l'intersection sur l'union (IoU, pour *Intersection over Union*) des boîtes englobantes, comme le rapport entre l'aire de l'intersection et l'aire de l'union de deux boîtes englobantes (Figure 2.12). Ensuite, on redéfinit les catégories précédentes en prenant en compte l'IoU, en ne considérant que les prédictions et vérité-terrains d'une même classe :

- Vrai Positif (TP) : une prédiction est superposée à une vérité-terrain avec une IoU supérieure à un seuil S (en général, 50%, 75% ou 95%).
- Faux Positif (FP) : une prédiction est superposée à une vérité-terrain, mais avec une IoU inférieure au seuil S.
- Faux Négatif (FN) : une boîte englobante de la vérité-terrain n'est pas détectée (il n'y a pas de prédiction correcte d'IoU supérieure au seuil S).
- Vrai Négatif (TN) : ne s'applique pas pour la détection d'objets. En théorie, cela correspondrait à toutes les boîtes non détectées et ne correspondant pas à un objet, ce qui n'est pas définissable.

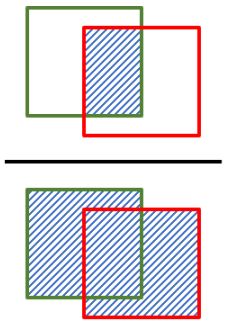


FIGURE 2.12 – Illustration de l'intersection sur l'union (IoU) entre une prédiction (carré rouge) et un objet de la vérité-terrain (carré vert). L'IoU correspond au rapport des aires hachurées.

Dans cette thèse, nous avons évalué la tâche de détection d'objets avec la mesure de précision moyenne (mAP, pour *mean Average Precision*) telle qu'elle est définie pour

18. https://github.com/rafaelpadilla/Object-Detection-Metrics

le *challenge* Pascal VOC (Everingham *et al.*, 2010). Nous décrivons donc ici le calcul de performance selon cette méthode. La mAP se calcule en deux étapes : le calcul de la précision pour chaque catégorie en considérant l'ensemble des images de test, puis la moyenne pour toutes les catégories. Pour chaque catégorie, les prédictions de chaque image sont donc catégorisées comme TP ou FP en fonction de leur superposition avec un objet de la vérité-terrain, après avoir éliminé les redondances avec une étape de « Suppression des Non Maxima » (NMS). Elles sont ensuite triées par score de confiance décroissant, toutes images confondues, afin d'être intégrées au calcul de la précision et du rappel de façon itérative ; cela permet de tracer une courbe de la précision en fonction du rappel (courbe Précision-Rappel, ou P-R). La première détection, de score de confiance le plus élevé, est probablement correcte, ce qui correspond à un TP. La précision associée est de 1, car il y a une détection correcte pour 1 détection au total. Si cette détection est incorrecte, alors la précision vaut 0. Le rappel commence lui toujours très proche de 0 : il y a soit 1 détection correcte, soit aucune, pour tous les objets de la vérité-terrain. La précision et le rappel vont ainsi être mis à jour pour chaque nouvelle détection considérée : si la détection est correcte (TP), alors la précision et le rappel augmentent légèrement ; si la détection est incorrecte, la précision diminue et le rappel n'est pas impacté. Le résultat est une courbe P-R en dents de scie comme celle de l'exemple sur la Figure 2.13. La performance est ensuite calculée comme l'aire sous la courbe P-R. Celle-ci est approximée de deux façons possibles :

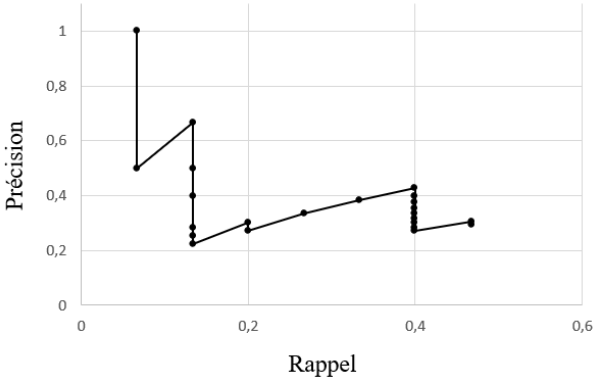
- Interpolation avec 11 points : la précision moyenne est la moyenne des précisions correspondant à 11 valeurs de rappel, régulièrement espacées de 0 à 1. Cela correspond à la moyenne des points rouges de la Figure 2.13b.
- Interpolation en tout point : la précision est approximée pour toutes les valeurs de rappel, en définissant une succession d'aires rectangulaires, qui absorbent les pics de la courbe. Cela correspond à l'aire sous la courbe en pointillés rouges de la Figure 2.13c.

Enfin, les précisions obtenues pour chaque catégorie d'objets sont moyennées afin d'obtenir la valeur finale de mAP, toujours entre 0 et 1.

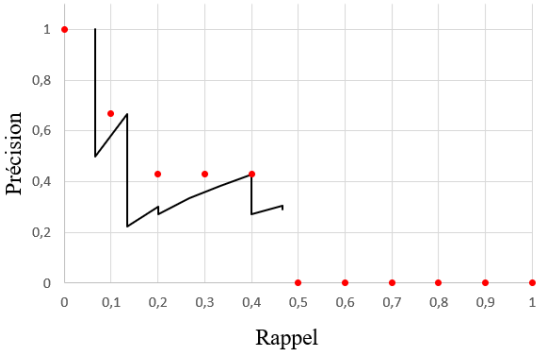
2.2.4 Données d'apprentissage et augmentation de données

Comme indiqué précédemment, un jeu de données dédié à l'apprentissage d'un réseau de neurones est décomposé en plusieurs sous-ensembles : entraînement, validation, et test. En fonction du nombre total d'exemples disponibles, le jeu d'entraînement peut correspondre à 70% des données, et jusqu'à 95%. Ensuite, les données de validation permettent de choisir les hyperparamètres permettant d'obtenir la meilleure performance. Et enfin, les données de test permettent de calculer la performance que l'on peut obtenir en conditions réelles, pour des données n'ayant pas servi à l'optimisation. En fonction de l'application, cette répartition peut se faire une seule fois de façon aléatoire, plusieurs fois afin de moyenner les résultats, ou de façon manuelle pour répartir les classes de façon équilibrée par exemple.

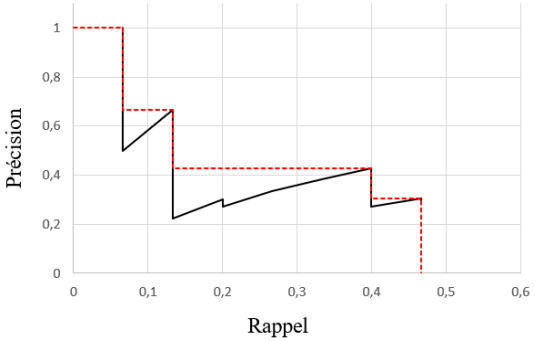
A chaque *epoch*, le réseau est confronté à toutes les données d'entraînement, avant que celles-ci ne soient mélangées et réparties dans de nouveaux *batches*. Cependant, il y a un risque de sur-apprentissage si ce sont les mêmes images qui sont présentées au réseau durant des dizaines, voire centaines d'*epochs*. Le principe d'« augmentation des données » a été proposé afin de limiter ce problème. Augmenter un jeu de données signifie créer de nouvelles données, en modifiant les données existantes (Montserrat *et al.*, 2017; Shorten



(a) Courbe précision-rappel pour une classe.



(b) Méthode 1 : interpolation 11 points.



(c) Méthode 2 : interpolation en tout point.

FIGURE 2.13 – Calcul de la précision moyenne (AP) pour une classe comme l'aire sous la courbe précision-rappel. Exemple avec deux méthodes d'interpolation.

et Khoshgoftaar, 2019). Lorsque l'on travaille avec des images, l'augmentation minimale qui est systématiquement appliquée correspond à des altérations des couleurs des pixels et du contraste, à l'application de bruit ou de légères rotations, ou encore au découpage des bords de l'image afin de décaler son contenu de quelques pixels. Des exemples de ces augmentations sont montrés sur la Figure 2.14. Cela permet à un réseau de neurones d'être confronté à des variations qui n'impactent pas le résultat attendu : l'intensité des couleurs ou la présence de bruit dans une image ne doit pas impacter la catégorie prédite.

(a) Image originale.

(b) Flou de moyenne avec un noyau de taille 7×7 pixels.

(c) Amélioration locale du contraste par égalisation d'histogramme (CLAHE).

(d) Retournement horizontal, suivi de flou de mouvement.

FIGURE 2.14 – Exemples d'augmentations géométriques et colorimétriques, obtenues avec la bibliothèque logicielle Albumentations [19].

Lorsque l'on ne dispose que de très peu d'images, alors ces images altérées peuvent être ajoutées au jeu de données initial. C'est ce que fait la méthode « mixup » notamment, en ajoutant des exemples virtuels par interpolation d'exemples réels du jeu de données (Zhang *et al.*, 2017). Ainsi, des opérations d'augmentation plus avancées ont été proposées dans la littérature et appliquées aux images ; quelques-unes de ces méthodes sont illustrées sur la Figure 2.15. Par exemple, le mélange de plusieurs images (SamplePairing (Inoue, 2018), MixedExample (Summers et Dinneen, 2019), RICAP (Takahashi *et al.*, 2019), CutMix (Yun *et al.*, 2019), AugMix (Hendrycks *et al.*, 2019)), ou l'effacement de pixels (DeVries et Taylor, 2017; Zhong *et al.*, 2020), sont des pratiques qui ne produisent pas des images réalistes pour l'oeil humain, mais qui ont montré de bons résultats pour les réseaux de neurones. Cependant, la plupart de ces méthodes ne s'appliquent aux images que pour une tâche de classification, ce qui a mené au développement de méthodes spécifiques à la détection d'objets (Zoph *et al.*, 2020) (Figure 2.15c) ou à la segmentation (ClassMix (Olsson *et al.*, 2021), Figure 2.15d). Enfin, des méthodes ont été proposées pour apprendre la meilleure séquence d'opérations à appliquer (AutoAugment (Cubuk *et al.*, 2019), RandAugment (Cubuk *et al.*, 2020), Faster AutoAugment (Hataya *et al.*, 2020),(Zoph *et al.*, 2020)). Malgré le grand nombre de méthodes proposées dans la littérature, celles-ci sont définies uniquement pour les images RGB, délaissant les autres types d'images. Nous abordons, dans la section 2.4.4, les approches dédiées aux cartes de profondeur et les méthodes d'augmentations associées.

2.3 Architectures de l'état de l'art

Dans cette section, nous décrivons quelques architectures de réseaux de neurones de la littérature. Parmi l'abondance de réseaux existants, nous avons choisi ceux-ci pour leur importance dans nos travaux, ou parce qu'ils font figure de référence pour l'apprentissage profond.

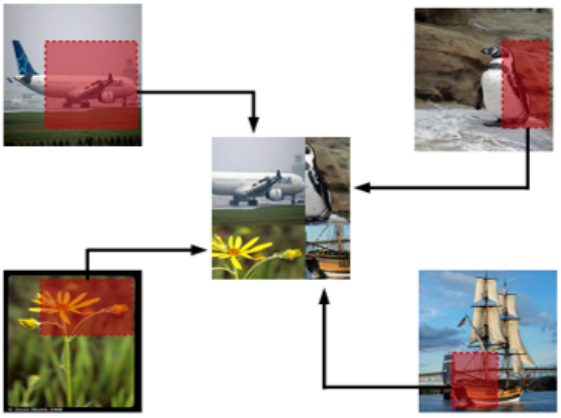
2.3.1 Classification d'images

2.3.1.1 AlexNet

La publication de l'immense jeu de données ImageNet (Deng *et al.*, 2009) en 2009, avec ses 15 millions d'images annotées et 22 000 catégories (dont seulement 1,2 millions d'images et 1000 catégories sont sélectionnées pour le *challenge* associé), a poussé la communauté scientifique à chercher des modèles de plus grande capacité que les modèles d'apprentissage classique utilisés jusque-là. En 2012, c'est vers les réseaux de neurones à convolution que se sont tournés les auteurs de Krizhevsky *et al.* (2012) afin de répondre à ce *challenge*, ouvrant la porte aux centaines d'architectures proposées par la suite. Le réseau AlexNet, nommé en hommage au premier auteur de la publication Alex Krizhevsky, est reconnu comme celui ayant permis la « redécouverte » de l'apprentissage profond, délaissé depuis les années 90. Depuis, l'engouement pour les réseaux de neurones n'a pas diminué ; l'augmentation des capacités des ordinateurs a permis l'entraînement de réseaux toujours plus gros et plus gourmands en temps de calcul, mais aussi plus performants.

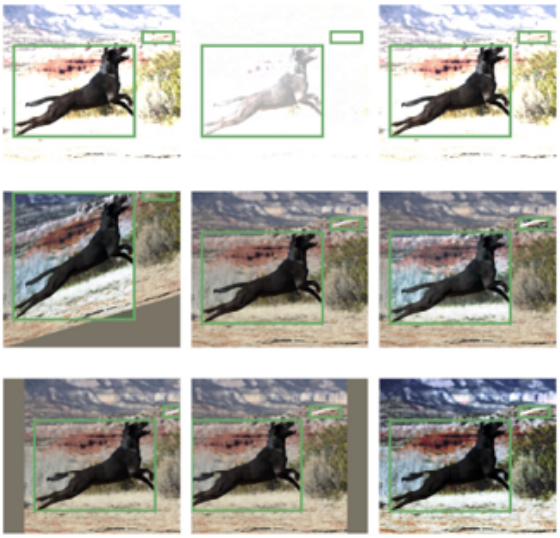
AlexNet est composé de 5 couches de convolution, suivies de 3 couches complètement

19. `https://albumentations.ai/docs/`

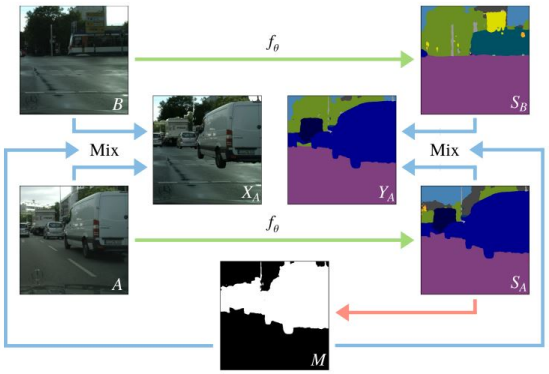


(a) Juxtaposition de patches avec la méthode RICAP (Takahashi *et al.*, 2019).

(b) Plusieurs méthodes de MixedExamples (Summers et Dinneen, 2019).



(c) Augmentations pour la détection d'objets (Zoph *et al.*, 2020).



(d) Augmentation pour la segmentation d'objets ClassMix (Olsson *et al.*, 2021).

FIGURE 2.15 – Exemples de méthodes d'augmentation d'images de la littérature.

connectées, comme le montre la Figure 2.16. Une importante contribution des auteurs est l'utilisation de la fonction non-linéaire ReLU (Nair et Hinton, 2010) comme fonction d'activation, ce qui a fortement réduit le nombre d'*epochs* nécessaires pour l'entraînement du réseau par rapport à la fonction tangente hyperbolique *tanh*. Ensuite, aussi petit soit ce réseau, son entraînement nécessitait plus d'espace mémoire que ne pouvait fournir un GPU de l'époque (soit 3Go). Les auteurs se sont donc appuyés sur la parallélisation de l'architecture entre plusieurs GPUs, une pratique qui s'est développée par la suite et est encore utilisée aujourd'hui malgré l'augmentation de la mémoire et vitesse de calcul de ces derniers. Concrètement, chaque couche du réseau proposé est décomposée en deux parties, exécutées chacune sur un GPU. Certaines couches sont synchronisées en permettant la communication entre les deux GPUs ; en d'autres termes, certains noyaux d'une couche ne reçoivent comme entrée que certaines couches de caractéristiques de la couche précédente, celles présentes sur le même GPU, alors que d'autres les reçoivent toutes.

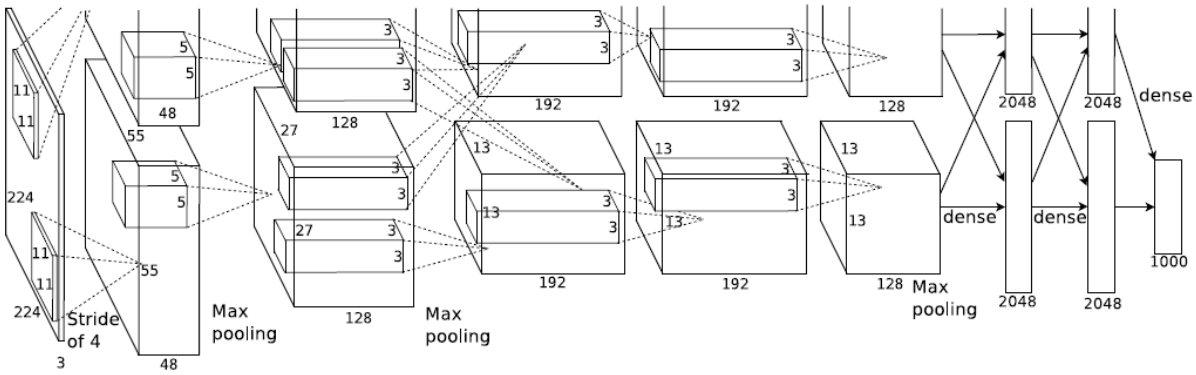


FIGURE 2.16 – Architecture du réseau AlexNet (Krizhevsky *et al.*, 2012).

2.3.1.2 Les modèles VGG

Plus tard, l'impact de la profondeur des réseaux à convolution a été investiguée par Simonyan et Zisserman (2014), toujours dans l'optique d'améliorer la performance de classification du jeu de données ImageNet. Ces travaux ont mené à la définition du réseau VGGNet, une architecture de référence dans la littérature, utilisée encore aujourd'hui pour l'extraction de caractéristiques. Plus précisément, il existe plusieurs versions de l'architecture VGG, comprenant respectivement 11, 13, 16 et 19 couches (dont les 3 dernières sont des couches complètement connectées). Les expériences menées ont montré une meilleure performance avec le réseau « le plus profond », c'est-à-dire comprenant le plus de couches.

2.3.1.3 Les modèles ResNets

Les réseaux de plus en plus profonds qui ont été proposés par la suite ont rapidement fait face à un problème dû au grand nombre de couches accumulées : la performance obtenue semble saturer, puis se dégrader, avec le nombre de couches. Ce phénomène contre-intuitif est illustré sur la Figure 2.17 : un réseau avec un plus grand nombre de couches obtient une erreur de classification plus élevée, sur le jeu de données ImageNet (He *et al.*, 2016).

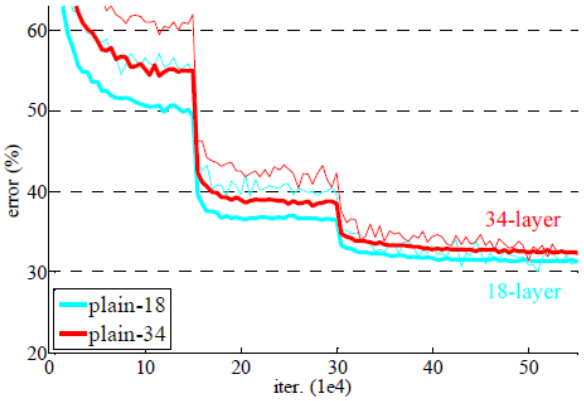


FIGURE 2.17 – Pourcentage d'erreur de classification sur l'ensemble d'entraînement (courbes fines) et de validation (courbes épaisses), pour deux réseaux de neurones à convolutions à 18 et 34 couches (inspirés de VGG). Image issue de He *et al.* (2016).

La solution qui a été étudiée, puis validée, consiste à utiliser des « raccourcis » jouant le rôle de la fonction d'identité. En effet, les auteurs sont partis du principe qu'un réseau plus profond doit être au moins aussi bon qu'un réseau moins profond. Les couches de convolution doivent donc permettre de ne pas transformer les cartes de caractéristiques en entrée lorsque cela n'est pas nécessaire. Or, cela demande d'apprendre à modéliser la fonction d'identité, en plus des caractéristiques spécifiques à la couche en question. Les « couches résiduelles » (Figure 2.18) proposées permettent de simuler par construction la fonction d'identité, sans que cela ne repose sur les poids de la couche. Formellement, si l'on souhaite qu'une couche modélise la transformation $H(x)$ entre une entrée x et une sortie y, alors les poids appris correspondent à la transformation $F(x) = H(x) - x$, et l'entrée x est ensuite ajoutée par concaténation. Ces couches résiduelles sont aussi appelées « blocs résiduels », puisqu'elles forment le bloc de base des nouvelles architectures que sont les ResNets. Avec un apprentissage facilité par ces blocs résiduels, les ResNets permettent d'accumuler un nombre de couches bien plus grand sans que la performance ne sature. Les architectures principales définies et couramment utilisées comportent 18, 34, 50, 101 et 151 couches. Ces réseaux bien plus profonds ont à nouveau obtenu les meilleurs résultats de classification pour le jeu de données ImageNet (He *et al.*, 2016), et ont remplacé les réseaux VGG dans la plupart des architectures. Par la suite, les blocs résiduels ont été intégrés à de nombreux autres réseaux, comme dans l'architecture MobileNet.

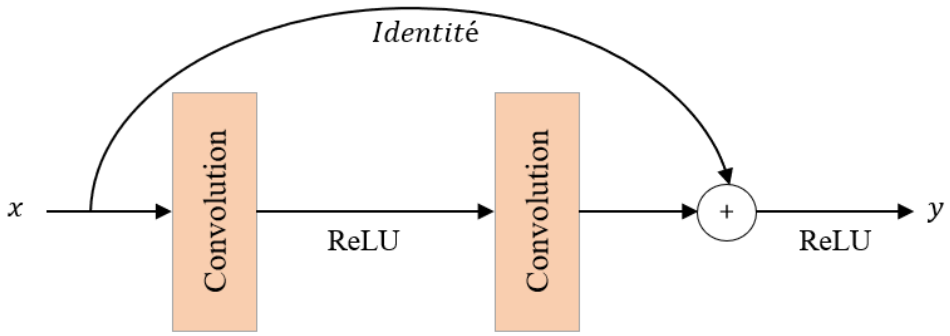


FIGURE 2.18 – Schéma d'une couche résiduelle (He *et al.*, 2016), sur laquelle sont basées les architectures ResNet.

2.3.1.4 Les modèles MobileNets

La famille des MobileNets a été introduite en 2017 (Howard *et al.*, 2017). Face au constat que les réseaux d'apprentissage profond s'améliorent au prix d'une taille et d'un temps d'inférence de plus en plus grands, les auteurs ont proposé une approche pour concevoir des réseaux plus petits, en se concentrant sur le temps d'inférence (aussi appelée latence), et la taille occupée par le réseau en mémoire. Ils ont donc créé une architecture qui s'adapte à une grande quantité d'applications, avec deux nouveaux hyperparamètres qui permettent à l'utilisateur d'ajuster l'architecture selon ses besoins : le multiplicateur de largeur et le multiplicateur de résolution. Les MobileNets s'appuient sur les convolutions séparables en profondeur, décrites dans la section 2.1.2. Celles des MobileNets ont une taille de noyau de 3×3 pixels (comme les convolutions standards de VGG), ce qui réduit fortement le nombre d'opérations nécessaires, pour une très légère perte de précision par rapport à des convolutions classiques. Nous détaillons ici les différentes architectures proposées pour les MobileNets, car c'est l'un des réseaux sur lesquels nous nous appuyons

dans cette thèse.

Architecture de MobileNetV1

Dans (Howard *et al.*, 2017), les auteurs définissent MobileNetV1, la première architecture basée sur le concept des MobileNets. La première couche est une convolution standard, puis le reste est une succession de couches de convolution séparables en profondeur. Toutes les couches sont suivies d'une couche BatchNorm et d'une fonction d'activation ReLU, sauf la dernière couche complètement connectée qui donne les caractéristiques à une couche *softmax*, pour la classification. L'architecture est donnée dans le Tableau 2.2.

TABLEAU 2.2 – Architecture de MobileNetV1 (Howard *et al.*, 2017). « Conv2d » est une convolution standard, « Conv depthwise separable » est une convolution séparable en profondeur, « n » est le nombre de répétitions de ce bloc, « AvgPool » est une couche de *pooling* par moyenne, « FC » est une couche complètement connectée, « k » est le nombre de classes.

Entrée	**Bloc**	**Canaux de sortie**	***Stride***	**n**
$224^2 \times 3$	Conv2d	32	2	1
$112^2 \times 32$	Conv2d depthwise separable	64	1	1
$112^2 \times 64$	Conv2d depthwise separable	128	2	1
$56^2 \times 128$	Conv2d depthwise separable	128	1	1
$56^2 \times 128$	Conv2d depthwise separable	256	2	1
$28^2 \times 256$	Conv2d depthwise separable	256	1	1
$28^2 \times 256$	Conv2d depthwise separable	512	2	1
$14^2 \times 512$	Conv2d depthwise separable	512	1	5
$14^2 \times 512$	Conv2d depthwise separable	1024	2	1
$7^2 \times 1024$	Conv2d depthwise separable	1024	1	1
$7^2 \times 1024$	AvgPool 7×7	-	-	1
$1^2 \times 1024$	FC + *Softmax*	k	1	1

Multiplicateur de largeur

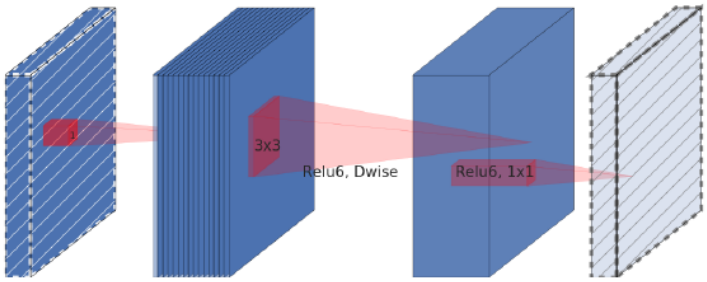
Pour réduire encore la taille du réseau et augmenter la vitesse d'inférence, les auteurs ont proposé d'ajouter un paramètre supplémentaire, dénoté α, nommé *width multiplier*. Ce coefficient s'applique au nombre de canaux en entrée et sortie de chaque couche : le nombre de canaux d'entrée M devient αM, et le nombre de canaux de sorties N devient αN, pour toutes les couches. α prend sa valeur entre 0 et 1, avec comme valeurs standards [1 ; 0,75 ; 0,5 ; 0,25]. La valeur de 1 est celle de référence, correspondant à MobileNetV1, les autres valeurs correspondent à des MobileNets réduits.

Multiplicateur de définition

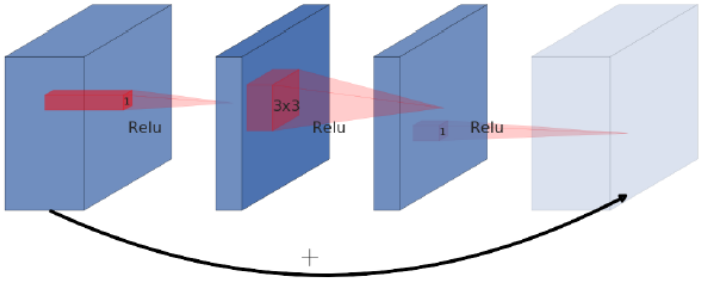
Le second hyperparamètre permet de réduire la taille des représentations apprises par le réseau, et donc le coût de calcul. Ce coefficient, dénoté ρ, est appliqué à l'image en entrée du réseau, ce qui réduit d'autant chaque couche intermédiaire. Dans la pratique, cela correspond à réduire la taille de l'image en entrée du réseau. Les valeurs habituellement utilisées comme taille d'entrée sont 224, 192, 160 et 128 pixels de côté, pour des images carrées.

Linear bottlenecks

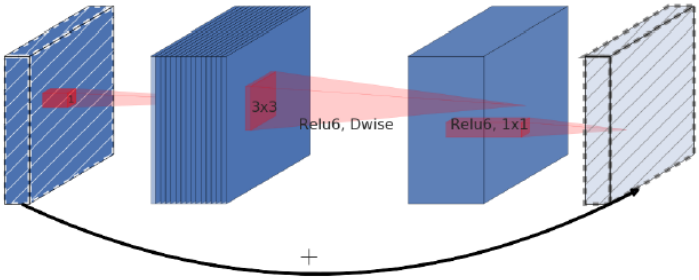
Poursuivant leur travaux autour des réseaux de convolution plus petits et plus rapides, les chercheurs à l'origine des MobileNets ont proposé en 2018 une amélioration à leur architecture, en introduisant les blocs *inverted residual with linear bottleneck* (Sandler *et al.*, 2018), que nous allons expliquer. Dans un réseau de neurones à convolution constitué de plusieurs blocs, les couches d'activations forment un ensemble de « variétés d'intérêt » (*manifolds of interest* dans le vocabulaire original). Dans le domaine de l'apprentissage profond, il est supposé que l'information est encodée dans une variété, qui se trouve elle-même dans un sous-espace de faible dimension. Le multiplicateur de largeur de MobileNetV1 est une façon de réduire la dimensionalité des variétés d'intérêt. Cependant, une analyse fine de la transformation des variétés à travers la fonction non-linéaire ReLU indique deux points importants : la transformation des variétés entre l'entrée et la sortie d'un bloc de convolution est linéaire, et toute l'information peut être préservée si elle est contenue dans un sous-espace de faible dimension de l'espace d'entrée.



(a) Goulots d'étranglement linéaires avec expansion.



(b) Bloc résiduel.



(c) Bloc résiduel inversé.

FIGURE 2.19 – Illustration d'un goulot d'étranglement linéaire (*linear bottleneck*) avec couche d'expansion, d'un bloc résiduel (tel que dans ResNet) et d'un bloc résiduel inversé (bloc de base de MobileNetV2). « Dwise » est une convolution en profondeur. Les couches hachurées ne comportent pas de non-linéarité ; les couches plus claires correspondent au bloc suivant. Images de Sandler *et al.* (2018).

Ces deux propriétés indiquent qu'il est possible d'optimiser l'architecture des MobileNets en insérant des goulots d'étranglement linéaires (*linear bottlenecks*) dans les blocs de convolution, afin de limiter la perte d'information due aux non-linéarités. Ils sont illustrés sur la Figure 2.19a. Ces goulots sont caractérisés par un ratio (ou facteur) d'expansion, correspondant au ratio entre la taille d'entrée et la taille interne au goulot d'étranglement. Cela revient donc à prendre en entrée du bloc une représentation compressée, de dimension réduite, puis de l'étendre à une représentation de grande dimension, et de la filtrer avec une convolution en profondeur. Enfin, les caractéristiques sont à nouveau projetées dans l'espace de faible dimension initial avec une convolution ponctuelle sans non-linéarité. Ce nouveau bloc de base réduit fortement la taille mémoire requise pour stocker les poids du réseau de neurones.

Inverted residuals

Les *bottlenecks* ainsi formés rappellent l'idée des blocs résiduels, dans lesquels l'entrée passe par plusieurs goulots d'étranglement, suivis d'une expansion (He *et al.*, 2016). Contrairement aux blocs résiduels, les blocs des MobileNets utilisent des liaisons directement entre les *bottlenecks*, comme indiqué sur les Figures 2.19b et 2.19c, ce qui leur vaut le nom de « blocs résiduels inversés ». Cette modification permet de diminuer la place occupée en mémoire, tout en fonctionnant légèrement mieux. Le bloc de base de MobileNetV1, la convolution séparable en profondeur, est maintenant remplacé par un *bottleneck* avec couche d'expansion et résidus inversé dans MobileNetV2, ce qui correspond à : une couche de convolution ponctuelle avec un facteur d'expansion t, une convolution en profondeur de noyau 3×3, puis une seconde couche de convolution ponctuelle pour obtenir la profondeur souhaitée en sortie du bloc (couche de projection), le tout avec une connexion résiduelle. Les opérations de ce bloc sont détaillées dans le Tableau 2.3, avec les dimensions d'entrée et de sortie de chacune des couches. Lorsque la couche interne de ce bloc a une profondeur de 0, le bloc correspond à la fonction d'identité, grâce à la connexion résiduelle. Lorsque le ratio d'expansion est inférieur à 1, le bloc correspond à un bloc résiduel classique. Dans le cas de MobileNetV2, le ratio d'expansion est choisi supérieur à 1.

TABLEAU 2.3 – Décomposition d'un bloc de base de MobileNetV2 (*bottleneck residual block*), avec un *stride* s et un facteur d'expansion t. Table publiée dans Sandler *et al.* (2018).

Entrée	**Bloc**	**Canaux de sortie**
$h \times w \times k$	Conv2d 1×1, ReLU6	$h \times w \times (tk)$
$h \times w \times (tk)$	Conv2d depthwise separable 3×3, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times (tk)$	Conv2d linéaire 1×1, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times k'$

Architecture de MobileNetV2

Comme MobileNetV1, le réseau débute par une couche de convolution classique, comportant 32 filtres. Ensuite, une série de 19 blocs *bottlenecks* avec résidus inversés forme le corps du réseau. Le facteur d'expansion est de 6, choisi expérimentalement. A nouveau, différentes versions de l'architecture sont proposées en faisant varier les multiplicateurs

de largeur et de définition. L'architecture par défaut est détaillée dans le Tableau 2.4. La fonction d'activation ReLU a été remplacée par ReLU6, son équivalent qui limite la valeur maximum de la fonction à la valeur 6, améliorant ainsi la robustesse pour le calcul en basse précision.

TABLEAU 2.4 – Architecture de MobileNetV2 (Sandler *et al.*, 2018). « Conv2d » est une convolution standard, « Bottleneck » est un bloc de base avec résidus inversés, « t » est le facteur d'expansion, « n » est le nombre de répétitions de ce bloc, « AvgPool » est une couche de pooling par moyenne, « k » est le nombre de classes. Le *stride* s'applique à la première couche de chaque bloc, les autres sont de 1.

Entrée	**Bloc**	t	**Canaux de sortie**	***Stride***	**n**
$224^2 \times 3$	Conv2d	-	32	2	1
$112^2 \times 32$	*Bottleneck*	1	16	1	1
$112^2 \times 16$	*Bottleneck*	6	24	2	1
$56^2 \times 24$	*Bottleneck*	6	32	1	1
$28^2 \times 32$	*Bottleneck*	6	64	2	1
$14^2 \times 64$	*Bottleneck*	6	96	1	1
$14^2 \times 96$	*Bottleneck*	6	160	2	1
$7^2 \times 160$	*Bottleneck*	6	320	1	5
$7^2 \times 320$	Conv2d 1×1	-	1280	2	1
$7^2 \times 1280$	AvgPool 7×7	-	-	1	1
$1^2 \times 1280$	Conv2d 1×1	-	k	1	1

Recherche d'architecture neuronale automatique : MnasNet

La recherche de l'architecture optimale est une tâche fastidieuse, que les chercheurs ont tenté d'optimiser au maximum. C'est dans ce contexte que sont apparus les algorithmes de recherche d'architecture neuronale, ou NAS en anglais pour *Neural Architecture Search*. Ces algorithmes permettent de faire des choix d'architecture basés sur l'optimisation d'un ensemble de métriques, comme le temps d'inférence ou le nombre de neurones par exemple, ce qui évite une recherche manuelle du nombre de couches, de neurones par couche, ou encore du type d'activation. Ces méthodes permettent un ajustement précis du compromis entre la taille du réseau ou son temps d'inférence, et sa performance.

Dans Tan *et al.* (2019), la mesure de latence est effectuée directement sur un appareil mobile, pour plus de précision. Alors que les approches précédentes se contentent d'explorer le nombre de blocs identiques à enchaîner pour créer une architecture complète, les auteurs cherchent ici à créer des blocs uniques et différents ; ils s'appuient sur l'intuition que les premières couches du réseau n'ont pas les mêmes besoins que les couches ultérieures. Ils proposent d'utiliser une convolution séparable en profondeur comme bloc de base, et définissent un certain nombre de blocs uniques, réduisant graduellement la définition d'entrée et augmentant la largeur des filtres. Chaque bloc est composé de couches identiques, dans lesquelles les opérations et connexions sont déterminées par recherche dans des sous-espaces. Par exemple, les convolutions peuvent être des convolutions normales, des convolutions en profondeur, ou des *bottlenecks* avec résidus inversés ; les noyaux sont de taille 3×3 ou 5×5 ; le nombre de canaux en sortie de chaque bloc est variable, etc.

En partant de l'architecture MobileNetV2 comme référence, l'architecture MnasNet-

A1 est définie grâce à un apprentissage par renforcement qui maximise une fonction de récompense. Cette récompense augmente avec la précision du modèle sur une tâche de classification, tout en respectant la contrainte d'un temps d'inférence inférieur à une certaine valeur. Le résultat est un ensemble de petits modules d'attention, basé sur les modules « Squeeze-and-Excite » (SE) proposés dans Hu *et al.* (2020) mais appliqués différemment dans la couche résiduelle du *bottleneck*.

Ces travaux autour de la recherche automatique d'architecture ont mené à l'insertion d'un module SE dans l'architecture de MobileNetV2, ce qui a mené à la définition d'une nouvelle version, MobileNetV3. Dans ce réseau, le bloc de base intègre le module SE entre la couche d'expansion et la couche de projection, comme représenté sur la Figure 2.20.

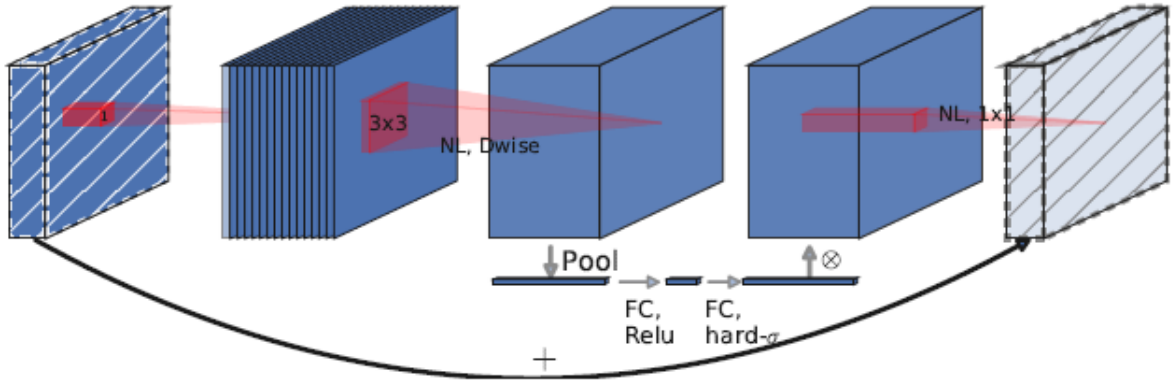


FIGURE 2.20 – Schéma d'un bloc de base de MobileNetV3 : *bottleneck* avec résidu inversé et couche Squeeze-and-Excite (SE) (Howard *et al.*, 2019).

Architecture de MobileNetV3

L'architecture finale de MobileNetV3 a été déterminée en utilisant un algorithme similaire à celui de MnasNet pour déterminer la structure globale du réseau, puis NetAdapt (Yang *et al.*, 2018) pour chercher le nombre optimal de filtres par couche. Ensuite, d'autres modifications ont été opérées pour réduire le nombre d'opérations de certaines couches, en début et fin d'architecture, comme la réduction de la taille de certains noyaux, ce qui permet également d'éliminer certaines couches devenues obsolètes, et la réduction du nombre de filtres pour limiter leur redondance. Ensuite, les fonctions d'activation ReLU de certaines couches d'activation ont été remplacées par des non-linéarités *swish* :

$$swish(x) = x \cdot \sigma(x) \tag{2.7}$$

La fonction sigmoïde σ, gourmande en calculs, est remplacée par son équivalent linéaire par morceaux (*hard sigmoid*), ce qui impacte également la fonction *swish* en la remplaçant par son équivalent *hard swish*.

$$h\text{-}swish(x) = x\frac{ReLU6(x+3)}{6} \tag{2.8}$$

Enfin, MobileNetV3 est proposé en deux tailles, *Large* pour une meilleure précision (Tableau 2.5) et *Small* pour une latence réduite (Tableau 2.6). Dans ces deux tableaux, « exp » correspond à la taille de la couche d'expansion, « SE » à la présence ou non d'un module « Squeeze-and-Excite », « NL » au type de non-linéarité (HS pour *hard swish*, et RE pour ReLU).

TABLEAU 2.5 – Architecture de MobileNetV3 *Large* (Howard *et al.*, 2019). NBN signifie l'absence de normalisation par *batch* (« no batch normalization ») entre la convolution et la non-linéarité (NL).

Entrée	**Bloc**	**exp**	**Canaux de sortie**	**SE**	**NL**	***Stride***
$224^2 \times 3$	Conv2d	-	16	-	HS	2
$112^2 \times 16$	*Bottleneck*, 3×3	16	16	-	RE	1
$112^2 \times 16$	*Bottleneck*, 3×3	64	24	-	RE	2
$56^2 \times 24$	*Bottleneck*, 3×3	72	24	-	RE	1
$56^2 \times 24$	*Bottleneck*, 5×5	72	40	✓	RE	2
$28^2 \times 40$	*Bottleneck*, 5×5	120	40	✓	RE	1
$28^2 \times 40$	*Bottleneck*, 5×5	120	40	✓	RE	1
$28^2 \times 40$	*Bottleneck*, 3×3	240	80	-	HS	2
$14^2 \times 80$	*Bottleneck*, 3×3	200	80	-	HS	1
$14^2 \times 80$	*Bottleneck*, 3×3	184	80	-	HS	1
$14^2 \times 80$	*Bottleneck*, 3×3	184	80	-	HS	1
$14^2 \times 80$	*Bottleneck*, 3×3	480	112	✓	HS	1
$14^2 \times 112$	*Bottleneck*, 3×3	672	112	✓	HS	1
$14^2 \times 112$	*Bottleneck*, 5×5	672	160	✓	HS	2
$7^2 \times 160$	*Bottleneck*, 5×5	960	160	✓	HS	1
$7^2 \times 160$	*Bottleneck*, 5×5	960	160	✓	HS	1
$7^2 \times 160$	Conv2d, 1×1	-	960	-	HS	1
$7^2 \times 960$	AvgPool 7×7	-	-	-	-	1
$1^2 \times 960$	Conv2d 1×1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	Conv2d 1×1, NBN	-	k	-	-	1

2.3.2 Détection d'objets

Initialement, les objets dans une image étaient détectés en faisant parcourir une fenêtre glissante sur l'image, et en classifiant les caractéristiques obtenues à chaque localisation. Les premières architectures neuronales de détection d'objets sont inspirées de cette approche : un premier réseau identifie des régions d'intérêt (notées RoI, pour *Region of Interest* en anglais), puis un second algorithme est appliqué à chacune de ces régions pour identifier la classe et affiner la position de la boîte englobante (réseau de neurones ou méthode classique). Ces méthodes sont nommées *double stage*, ou « en deux étapes », par opposition aux méthodes *single stage*, ou « en une étape », qui déterminent par régressions les coordonnées de la boîte englobante, tout en identifiant la classe avec un seul modèle, entraîné de bout en bout. L'architecture *double stage* la plus connue est R-CNN, à partir de laquelle plusieurs réseaux ont été proposés : Fast R-CNN, Faster R-CNN, puis Mask R-CNN (qui fait également la segmentation des objets dans l'image). Ces réseaux ont régulièrement obtenu la première place des *benchmarks* de détection d'objets et sont très prisés de la communauté scientifique, mais leur taille et durée d'inférence élevées rendent leur utilisation difficile dans un contexte mobile ou embarqué. Au contraire, les réseaux *single stage* sont plus compacts et rapides, mais leur capacité d'apprentissage réduite les rend moins performants. Ils sont donc privilégiés pour les applications embarquées dans lesquelles le temps de réponse est primordial et la mémoire limitée, alors que les réseaux *double stage* sont utilisés lorsque la précision est cruciale. Face à ces réseaux de plus en plus gourmands en données et en calculs, plusieurs travaux se sont penchés sur les détec-

TABLEAU 2.6 – Architecture de MobileNetV3 *Small* (Howard *et al.*, 2019). NBN signifie l'absence de normalisation par *batch* (« no batch normalization ») entre la convolution et la non-linéarité (NL).

Entrée	**Bloc**	**exp**	**Canaux de sortie**	**SE**	**NL**	***Stride***
$224^2 \times 3$	Conv2d	-	16	-	HS	2
$112^2 \times 16$	*Bottleneck*, 3×3	16	16	✓	RE	1
$56^2 \times 16$	*Bottleneck*, 3×3	72	24	-	RE	1
$28^2 \times 24$	*Bottleneck*, 3×3	88	24	-	RE	1
$28^2 \times 24$	*Bottleneck*, 5×5	96	40	✓	HS	1
$14^2 \times 40$	*Bottleneck*, 5×5	240	40	✓	HS	1
$14^2 \times 40$	*Bottleneck*, 5×5	240	40	✓	HS	1
$14^2 \times 40$	*Bottleneck*, 5×5	120	48	✓	HS	1
$14^2 \times 48$	*Bottleneck*, 5×5	144	48	✓	HS	1
$14^2 \times 48$	*Bottleneck*, 5×5	280	96	✓	HS	1
$7^2 \times 96$	*Bottleneck*, 5×5	576	96	✓	HS	1
$7^2 \times 96$	*Bottleneck*, 5×5	576	96	✓	HS	1
$7^2 \times 96$	Conv2d, 1×1	-	576	✓	HS	1
$7^2 \times 576$	AvgPool 7×7	-	-	-	-	1
$1^2 \times 576$	Conv2d 1×1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	Conv2d 1×1, NBN	-	k	-	-	1

teurs de petite taille qui ne nécessitent qu'une seule étape.

Dans les prochains paragraphes, nous présentons les architectures les plus utilisées pour la détection d'objets, en commençant par la méthode *double stage* la plus utilisée, la famille R-CNN. Ensuite, nous détaillons les réseaux *single stage* utilisés dans nos travaux : YOLO et SSD. Pour toutes ces architectures, nous décrivons les itérations successives ayant mené aux architectures les plus récentes.

2.3.2.1 La famille R-CNN

R-CNN

Face à la performance stagnante des méthodes de détection d'objets classiques appliquées au *benchmark* Pascal VOC, et au succès du réseau AlexNet pour le *challenge* ILSVRC, Girshick *et al.* (2016) se sont penchés sur l'utilisation d'un réseau de neurones à convolutions pour extraire des caractéristiques de l'image d'entrée, au lieu des caractéristiques manuelles utilisées jusque-là. Ils ont proposé la méthode R-CNN, pour *Regions with CNN features*, qui se compose des trois étapes suivantes (et illustrées sur la Figure 2.21) : extraction de régions d'intérêt (« *region proposals* »), extraction d'un vecteur de caractéristiques pour chaque région avec un réseau de neurones à convolution, et classification des caractéristiques avec un SVM (machine à vecteurs de supports, ou *Support Vector Machine* en anglais). Ils ont également testé et approuvé l'ajout d'un modèle de régression linéaire afin de préciser les coordonnées de la boîte englobante par rapport à la région d'intérêt. Cette approche leur a ainsi permis d'améliorer les résultats, mesurés par la mAP, de plus de 30% par rapport aux autres méthodes de l'époque. Ils ont aussi montré qu'une légère modification de l'architecture permettait de l'appliquer à la segmentation d'objets.

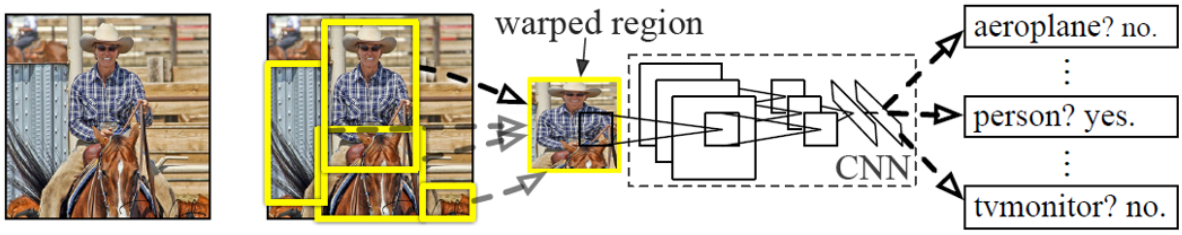


FIGURE 2.21 – Détection d'objets avec R-CNN. Figure modifiée de Girshick *et al.* (2016).

La méthode *selective search* (Uijlings *et al.*, 2013) est utilisée pour proposer des régions d'intérêt, environ 2000 par images. Elle peut cependant être remplacée par n'importe quel autre algorithme produisant des régions indépendantes de la catégorie d'objets. Ensuite, chaque région est redimensionnée par une opération appelée « *affine image warping* », une déformation affine qui permet de produire une imagette de taille 227×227 pixels à fournir au réseau de neurones. C'est ensuite AlexNet qui est utilisé afin de produire un vecteur de caractéristiques de dimension 4096, en éliminant sa dernière couche dédiée à la classification. Enfin, un modèle SVM par classe est entraîné pour classifier les vecteurs de caractéristiques. A l'inférence, chaque SVM produit un score, indiquant la probabilité qu'une région corresponde à cette classe. Une étape de suppression des non-maxima permet de réduire le nombre de régions en se basant sur leurs scores et leurs intersections.

Enfin, une autre contribution importante de Girshick *et al.* (2016) est l'utilisation du jeu de données ImageNet comme pré-entraînement supervisé. Jusque-là, les quelques réseaux de neurones proposés pour la détection étaient pré-entraînés sur une tâche de façon non-supervisée, afin de tirer parti d'une plus grande quantité de données même si elles ne sont pas annotées. Avec R-CNN, le réseau AlexNet est d'abord entraîné pour la classification d'images avec ImageNet, puis entraîné pour la détection avec les images et annotations de Pascal VOC. Cette étape de *fine-tuning* est réalisée en remplaçant la dernière couche d'AlexNet par une nouvelle couche avec le nombre correct de classes. Bien que les prédictions pourraient être tirées directement de cette couche sans utiliser de SVM, la performance est meilleure avec ces derniers car la localisation des objets obtenue est plus précise (Girshick *et al.*, 2014, *supplementary material*).

Fast R-CNN

R-CNN présente cependant un certain nombre de limitations : il est très lent à entraîner puisque chaque région passe à travers le réseau une par une, l'entraînement demande plusieurs étapes, et l'inférence est extrêmement lente (presque une minute par image). Des améliorations ont donc été proposées, dont le résultat est l'architecture Fast R-CNN : un entraînement et une inférence plus rapides, pour une meilleure performance (Ross Girshick, 2015).

Ces résultats ont d'abord été obtenus en remplaçant l'architecture AlexNet par VGG-16, un réseau plus profond. Ensuite, ce réseau prend en entrée à la fois des propositions de région et l'image d'origine. L'image est traitée par le réseau pour produire une carte de caractéristiques, puis une nouvelle couche appelée *RoI pooling* l'utilise pour générer un vecteur de caractéristiques pour chaque proposition de région. L'opération de *RoI pooling* correspond à une couche *max-pooling* appliquée séparément à chaque région proposée, qui sont de tailles variables, de façon à ce que la sortie ait une taille fixe. Pour une sortie souhaitée de taille $H \times W$ et une région de taille $h \times w$, la région est divisée en $h/H \times w/W$ sous-régions, pour lesquelles la valeur maximale est sélectionnée. La carte de

caractéristiques obtenue pour chaque région est alors « applatie » pour former un vecteur. Ce vecteur passe ensuite à travers une séquence de couches complètement connectées, puis à travers deux branches parallèles afin de produire deux sorties : les probabilités que la région contienne un objet de chaque classe, et des coordonnées de boîte englobante pour chaque classe.

Du point de vue de l'entraînement, l'étape d'optimisation demande beaucoup moins de temps et d'espace en mémoire. En éliminant le SVM, les caractéristiques de chaque région n'ont plus besoin d'être stockées en mémoire, et l'entraînement se fait de bout en bout, en une seule étape (sans compter l'étape de proposition de régions d'intérêt). En produisant une couche de caractéristiques pour l'image entière, un seul passage à travers le réseau d'extraction des caractéristiques est nécessaire pour toutes les régions d'une image, ce qui réduit à nouveau le nombre de valeurs intermédiaires à stocker pour la rétro-propagation.

Enfin, une analyse de la durée d'inférence à travers les différentes couches a indiqué que près de 40% de la propagation avant était passée dans les couches complètement connectées. Celles-ci ont donc été compressées par Décomposition en Valeurs Singulières (notée SVD, pour *Singular Value Decomposition* en anglais) (Denton *et al.*, 2014), ce qui a permis de remplacer une grande couche FC par deux plus petites. Dans l'étude présentée, la réduction du temps d'inférence pour une image est de 320ms à 223ms (voir Ross Girshick, 2015, Figure 2).

Faster R-CNN

Par la suite, Ren *et al.* (2017) se sont penchés sur l'étape de proposition de région, devenue le point bloquant de tout le processus de détection d'objet avec Fast R-CNN. La nouvelle évolution de l'architecture, dénommée Faster R-CNN, introduit un nouveau modèle dédié à la proposition de régions d'intérêt : RPN (*Region Proposal Network*). Ce réseau utilise les mêmes caractéristiques que le détecteur Fast R-CNN afin de prédire des régions et un score d'*objectness* pour chacune. L'*objectness* d'une région peut être définie comme l'appartenance de cette région à une catégorie d'objets, par opposition à l'arrière-plan. Les régions obtenues sont ensuite celles utilisées par Fast R-CNN afin d'effectuer les prédictions finales. En ce qui concerne l'entraînement, c'est d'abord le réseau RPN qui est entraîné de bout en bout, puis il est intégré à Fast R-CNN en réutilisant les couches de convolution pour la génération de caractéristiques. Ces couches servent comme « mécanisme d'attention » afin d'indiquer à Fast R-CNN les éléments à chercher dans l'image, puisqu'elles sont déjà entraînées. Le résultat final est un réseau capable de détecter des objets à une vitesse de 5 images par seconde sur un GPU, avec une meilleure performance que les autres réseaux de la littérature, et en ne générant que 300 propositions de région par image. Bien que très performant, ce réseau est toutefois trop lent pour des applications qui demandent une détection en temps réel, en particulier lorsqu'il n'y a pas de GPU disponible.

2.3.2.2 La famille YOLO

You Only Look Once (YOLO)

Dans Redmon *et al.* (2016), les auteurs proposent une nouvelle conception des réseaux de neurones pour la détection d'objets. Alors que les approches *double stage* comme R-CNN reviennent à appliquer un classifieur à différents endroits de l'image, ils considèrent ici la tâche de détection comme la régression de boîtes englobantes et de probabilités de classe directement depuis les images, en une seule étape et non plus à partir de régions d'intérêt extraites des images. C'est ce qui a inspiré le nom de ce réseau, You Only Look

Once : il n'est nécessaire de regarder qu'une seule fois l'image pour prédire les objets. Ce réseau peut donc être entraîné de bout en bout ; il est aussi extrêmement rapide, et bien plus performant que les réseaux qui pouvaient faire l'inférence en temps réel existant à ce moment-là. Cette performance est due au fait que YOLO considère l'image en entier, et non une petite fenêtre, incluant plus de contexte. De plus, les expériences menées montrent que YOLO apprend une représentation plus généralisable à d'autres domaines que R-CNN.

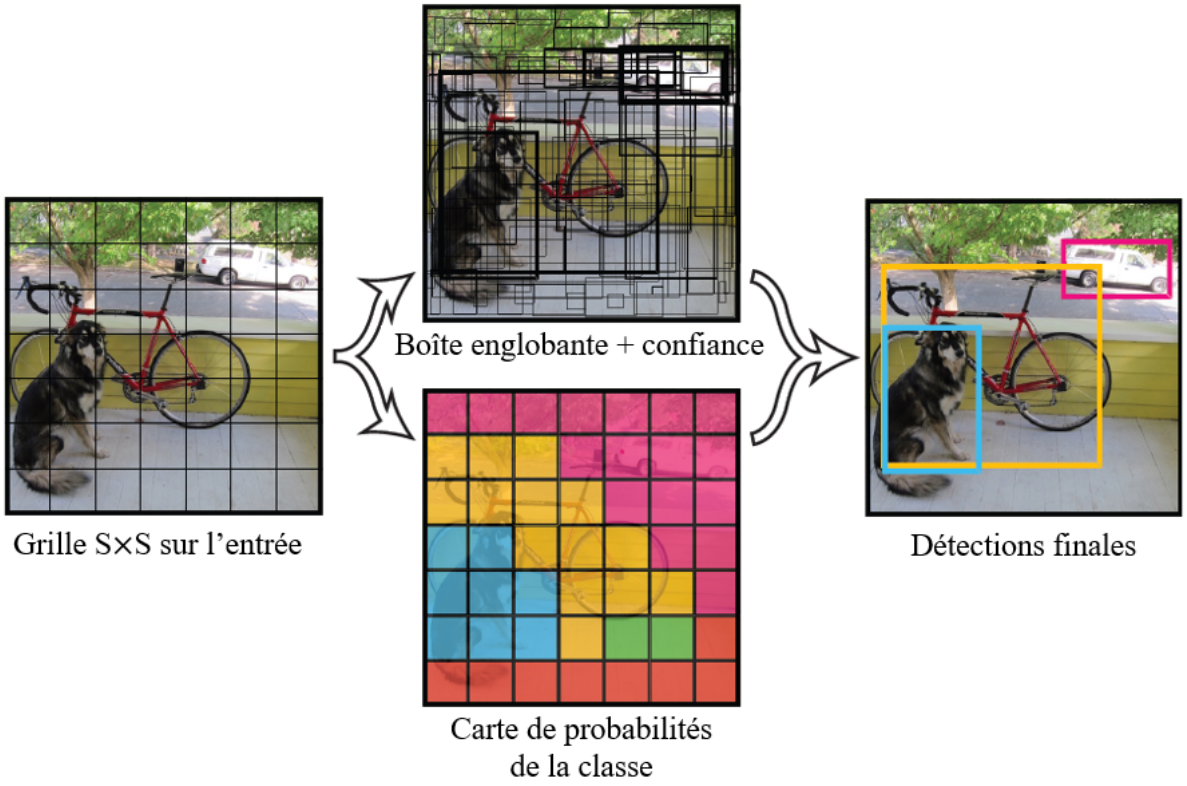


FIGURE 2.22 – Détection d'objets avec YOLO (Redmon *et al.*, 2016).

Pour obtenir ces résultats, YOLO divise l'image en entrée en une grille de cellules, de taille $S \times S$ pixels (Figure 2.22). Les objets sont associés à une cellule de la grille lorsque leur centre tombe dans cette cellule. Chaque cellule prédit un certain nombre de boîtes englobantes, noté B, avec un score de confiance pour chacune de ces boîtes. Ce score représente à la fois la confiance en la présence d'un objet, mais également la localisation de la boîte : il se calcule par l'intersection sur l'union (IoU) entre la boîte prédite et n'importe quelle boîte englobante de la vérité-terrain. Ainsi, s'il n'y a pas d'objet dans la cellule, le score doit être 0. YOLO prédit les coordonnées du centre de la boîte, relativement à la cellule associée, ainsi que sa hauteur et sa largeur, relativement à l'image. Ainsi, ces prédictions sont bornées entre 0 et 1, ce qui facilite l'optimisation de la fonction de perte choisie qui est la somme des résidus au carré. De plus, chaque cellule prédit les probabilités conditionnelles de chaque classe, valables pour toutes les boîtes englobantes prédites pour la cellule. A l'inférence, les probabilités conditionnelles sont multipliées aux scores de confiance de chaque boîte, ce qui donne un score de confiance pour chaque classe, et pour chaque boîte englobante prédite.

En ce qui concerne l'architecture neuronale (illustrée sur la Figure 2.23), YOLO est composé d'abord de 24 couches de convolution qui extraient les caractéristiques de l'image

en entrée, puis de 2 couches complètement connectées pour faire les prédictions. Bien qu'il soit inspiré du modèle GoogLeNet (Szegedy *et al.*, 2015), il n'utilise pas les mêmes modules Inception mais simplement des couches de réduction 1×1 suivies de couches de convolution 3×3. Afin de proposer un réseau encore plus rapide, les auteurs définissent également une version plus petite, dénommée Fast YOLO, avec seulement 9 couches de convolution au lieu de 24, et moins de filtres dans chacune de ces couches.

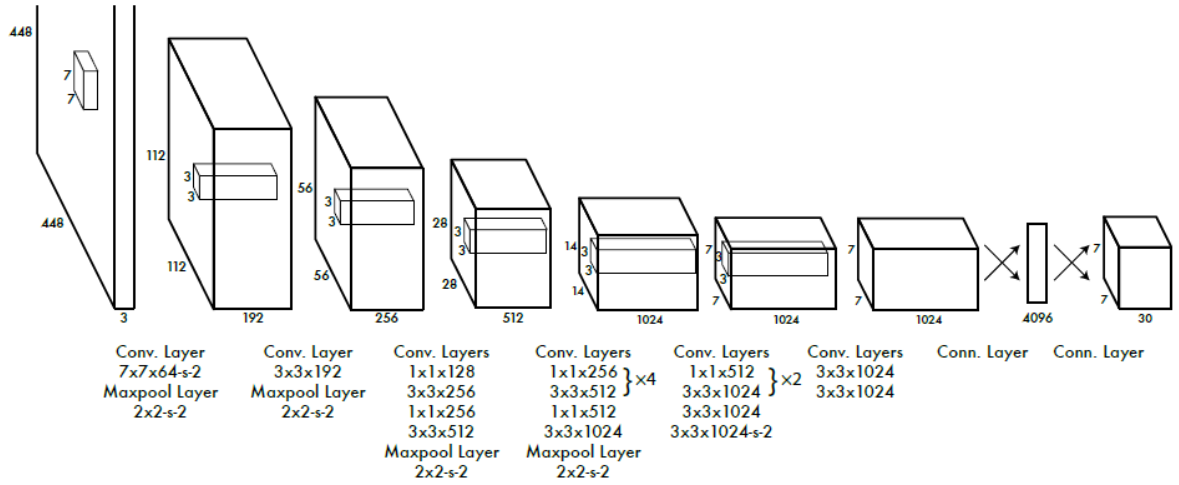


FIGURE 2.23 – Architecture du réseau YOLO (Redmon *et al.*, 2016).

Alors que plusieurs boîtes prédisent le même objet, un seul prédicteur est associé à chaque objet à l'entraînement, ce qui mène à des prédicteurs qui se spécialisent sur une classe, une taille, ou encore un rapport de forme. Pour la base de données Pascal VOC, YOLO prédit 98 boîtes englobantes par image, avec 2 boîtes englobantes par cellule. Bien que cela mène à ne prédire que très peu de faux positifs dans l'arrière-plan, ces deux boîtes par cellule, avec une seule classe possible pour chaque, limitent fortement la capacité de prédiction. De plus, les prédicteurs se spécialisant sur les objets de la vérité-terrain, le réseau est peu performant lorsque des objets de taille ou rapport de forme inhabituels sont présents dans les images de test. Enfin, contrairement à la plupart des réseaux de neurones qui sont implémentés avec les *frameworks* Tensorflow, Caffe ou PyTorch, YOLO est implémenté dans son propre *framework* Darknet (Redmon *et al.*, 2019).

YOLOv2

Dans une publication ultérieure, les mêmes auteurs proposent de nombreuses améliorations à la méthode YOLO, intégrées dans un nouveau réseau YOLOv2 (Redmon et Farhadi, 2017). Cette nouvelle architecture est à la fois plus rapide et plus performante que YOLO, mais également que les méthodes de l'état de l'art Faster R-CNN et SSD (dont elle s'inspire). Parmi les améliorations du réseau et sans entrer dans les détails, on compte : l'utilisation de couches de normalisation par *batch* (*Batch Normalization* en anglais) pour régulariser le réseau, ce qui permet d'enlever les couches *Dropout* ; le *fine-tuning* du classifieur à une définition plus élevée (448×448 pixels au lieu de 224×224) ; l'utilisation de boîtes pré-définies (les *anchor boxes*) pour prédire la localisation des objets dans les images ; la sélection automatique des dimensions des *anchor boxes* avec un algorithme de *clustering* k-*means* appliqué aux données d'entraînement ; une branche *passthrough* qui concatène des couches de caractéristiques de plus haute définition à celles en sortie du réseau, pour une meilleure détection des petits objets ; un entraînement à différentes

définitions, afin de rendre le réseau plus robuste et de faire varier facilement la vitesse d'inférence une fois le réseau entraîné ; un nouveau réseau de base, Darknet-19, inspiré de VGG-16 mais demandant bien moins d'opérations. En particulier, il est à noter que le réseau n'utilise plus de couches complètement connectées, ce qui lui permet de faire varier la taille de l'image en entrée. De plus, les nouvelles *anchor boxes* associent une classe par boîte prédite, et non une classe par localisation spatiale, ce qui permet de prédire un plus grand nombre d'objets, même s'ils se superposent.

Tous ces éléments ont permis d'augmenter le nombre d'objets prédits (de 98 à plus de 1000), d'améliorer la détection des petits objets, et de tous les objets de façon générale. Pour une performance équivalente, YOLOv2 détecte les objets du *benchmark* Pascal VOC à une vitesse de 90 images par seconde, alors que Fast R-CNN n'atteint que 0,5 image par seconde (voir Redmon et Farhadi, 2017, Table 2).

YOLO9000

Alors que la tâche de classification dispose de l'immense jeu de données ImageNet comme source d'entraînement, la détection d'objets ne dispose que d'un nombre limité de données annotées, avec les jeux Pascal VOC et MS COCO. Pour répondre à ce problème, les auteurs de Redmon et Farhadi (2017) ont proposé une méthode pour entraîner YOLOv2 à la fois sur une tâche de classification et de détection, ce qui leur permet de tirer parti d'ImageNet même sans annotations de détection. L'entraînement de YOLOv2 avec à la fois ImageNet et MS COCO a donné naissance au réseau YOLO9000, capable de détecter des objets de 9000 catégories différentes en temps réel, avec des annotations de détection pour seulement une partie d'entre elles.

Pour cela, ils utilisent une organisation hiérarchique des classes en suivant l'ontologie WordNet [20], qui permet de construire un arbre orienté *WordTree* comportant toutes les classes des bases de données à combiner. Ainsi, un objet associé à la classe « *jet* » sera également associée à la classe « *airplane* » (avion) par exemple, « jet » n'étant pas une classe du jeu de données de détection. La couche *softmax* en sortie du réseau, qui répartit les probabilités de chacune des classes, est remplacée par de multiples couches *softmax* recouvrant seulement un sous-ensemble des classes chacune, et suivant la hiérarchie de l'arbre construit. Pour chaque prédiction, le modèle prédit la boîte englobante, ainsi que l'arbre de probabilités. La classe finale est obtenue en parcourant cet arbre, en suivant le chemin de confiance la plus élevée, jusqu'à atteindre un seuil.

En ce qui concerne l'optimisation de ce réseau avec plusieurs bases de données, les auteurs proposent de différencier les images annotées pour la classification et pour la détection. Si une image possède une annotation de détection, alors le réseau est optimisé avec la fonction de coût complète. En revanche, si l'image correspond à de la classification et ne comporte pas de boîte englobante, alors seule la partie de la fonction de coût dédiée à la classification est utilisée, et la régression de la boîte englobante est ignorée. Cela permet au réseau d'apprendre à reconnaître des classes bien plus variées que celles présentes uniquement dans les bases de données de détection, même sans les annotations correspondantes.

YOLOv3

De nouvelles améliorations dans l'architecture de YOLO ont donné naissance au modèle YOLOv3 (Redmon et Farhadi, 2018), représenté sur la Figure 2.24. La couche *softmax* en sortie du réseau, qui prédit la classe de chaque prédiction, est remplacée par des classi-

20. https://wordnet.princeton.edu/

fieurs indépendants pour chaque classe, dans une logique de classification multi-classe. En effet, certains objets peuvent parfois correspondre à plusieurs classes, notamment dans des jeux de données plus complexes dont certaines catégories ne sont pas exclusives. Ensuite, YOLOv3 effectue des prédictions à 3 échelles différentes, en suivant la logique des *Feature Pyramid Networks* (FPN) : des prédictions sont effectuées à partir de couches de caractéristiques de différentes définitions. Les prédictions de la première échelle sont obtenues en ajoutant des couches de convolution à la suite de l'extracteur de caractéristiques ; la dernière de ces couches donne les prédictions pour la localisation de la boîte englobante, le score, ainsi que la probabilité de chaque classe. Pour la deuxième échelle, une couche de caractéristiques à la base de la première branche est extraite et sur-échantillonnée par deux pour augmenter sa définition spatiale. Une seconde couche de caractéristiques antérieure à la première branche est extraite, de même taille que la couche sur-échantillonnée, et elles sont concaténées afin d'obtenir à la fois des informations sémantiques et des détails fins. Plusieurs couches de convolution sont ensuite appliquées, et des prédictions sont à nouveau faites sur la couche de sortie. Enfin, des prédictions sont faites à une troisième échelle en répétant ce processus, ce qui permet de capitaliser sur les caractéristiques de haut niveau sémantique obtenues jusque-là, tout en bénéficiant des caractéristiques de haute définition spatiale provenant du début du réseau.

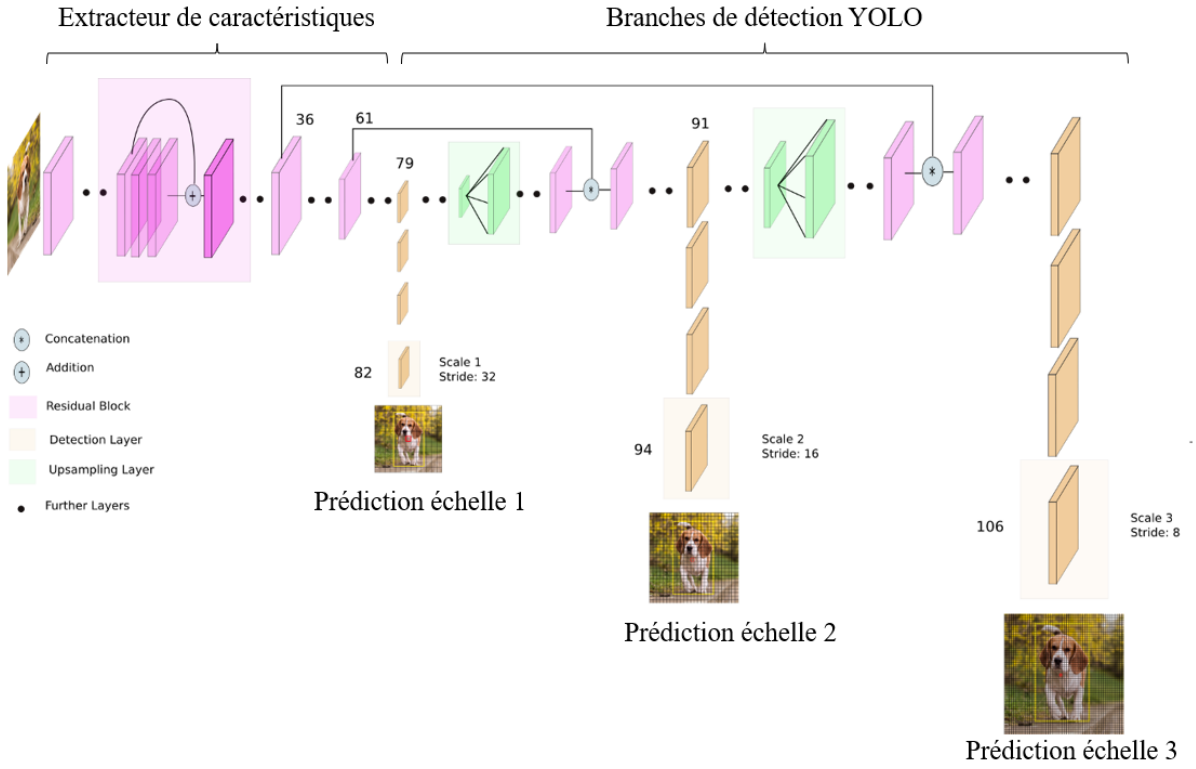


FIGURE 2.24 – Architecture du détecteur d'objets YOLOv3 [21]. Les nombres correspondent aux indices des couches.

A chaque échelle, les boîtes englobantes sont prédites comme indiqué sur la Figure 2.25. L'image est décomposée en une grille de régions, à chaque région est associée un ensemble B de boîtes englobantes par défaut. Pour chaque boîte par défaut, le réseau prédit le vecteur de coordonnées $t_* = (t_x, t_y, t_w, t_h, t_o)$, qui permet de calculer les coordonnées de

21. https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

la boîte englobante prédite avec les équations suivantes :

$$b_x = \sigma(t_x) + c_x \quad b_y = \sigma(t_y) + c_y \quad b_w = p_w e^{t_w} \quad b_h = p_h e^{t_h} \tag{2.9}$$

dans lesquelles (b_x, b_y) correspond aux coordonnées du centre de la boîte englobante prédite, et (b_w, b_h) aux dimensions de cette boîte prédite ; (c_x, c_y) sont les coordonnées du coin en haut à gauche de la région dans laquelle se situe la boîte par défaut, (p_w, p_h) sont les dimensions de la boîte par défaut ; σ correspond à la fonction sigmoïde, et e correspond à la fonction exponentielle. Le dernier élément du vecteur de prédiction, t_o, correspond au score d'objectivité (*objectness*), qui vaut 1 si la boîte prédite contient un objet, et 0 sinon. La fonction de perte pour la localisation des boîtes englobantes est la somme des carrés des résidus entre ce vecteur de prédiction t_* et la vérité terrain $\hat{t}_*$. Une seconde fonction de perte est utilisée pour la classification de chaque boîte englobante, correspondant à une fonction d'entropie croisée binaire pour chacune des classes.

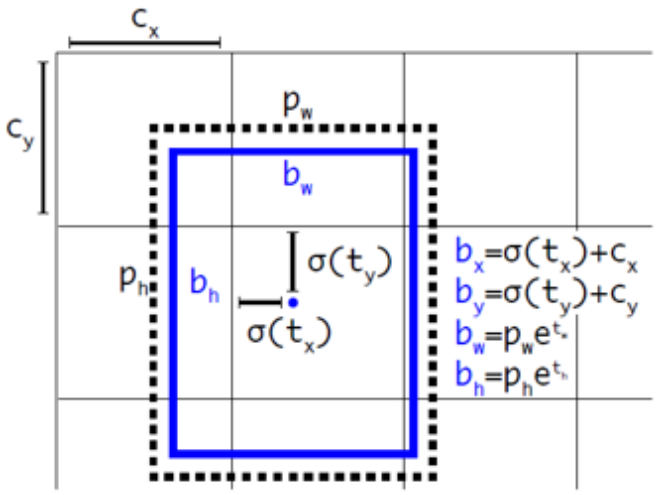


FIGURE 2.25 – Illustration de la prédiction des coordonnées d'une boîte englobante (représentée en bleu) pour YOLOv2 et YOLOv3, à partir d'une boîte par défaut (*prior*, représentée en pointillés). Illustration de Redmon et Farhadi (2017).

L'extracteur de caractéristiques est encore amélioré par rapport à YOLOv2, en s'inspirant de Darknet-19 et en y intégrant des couches résiduelles. Il est nommé Darknet-53, en référence aux 53 couches de convolution qui le composent. Malgré cette taille bien plus importante, il permet un plus grand nombre d'opérations par secondes, grâce à une utilisation plus efficace du GPU. Ainsi, YOLOv3 est un bon compétiteur dans l'état de l'art, avec des difficultés à localiser correctement les boîtes englobantes, mais bien plus rapide que les autres réseaux dits « de petite taille » (par rapport à un réseau tel que Faster R-CNN) tels que RetinaNet ou SSD, pour le jeu de données MS COCO.

2.3.2.3 L'architecture SSD

Le modèle Single Shot Detector (SSD) (Liu *et al.*, 2016) est contemporain à YOLO et également conçu pour la détection très rapide d'objets à plusieurs échelles dans des images, à partir de boîtes par défaut définies dans les cartes de caractéristiques. Ces boîtes par défaut couvrent plusieurs tailles et rapports de formes, et sont associées à un score de présence des objets pour chaque classe. Lorsque le réseau SSD fait une prédiction à

partir d'une image, toutes les boîtes par défaut donnent un score, ainsi que les ajustements nécessaires des bords de la boîte pour être au plus proche des objets identifiés. Cela permet d'éviter une étape de proposition de région, comme pour YOLO, et en fait un détecteur d'objets rapide et facile à entraîner de bout en bout. Les deux architectures sont comparées sur la Figure 2.26, notamment les dimensions des couches de caractéristiques de SSD.

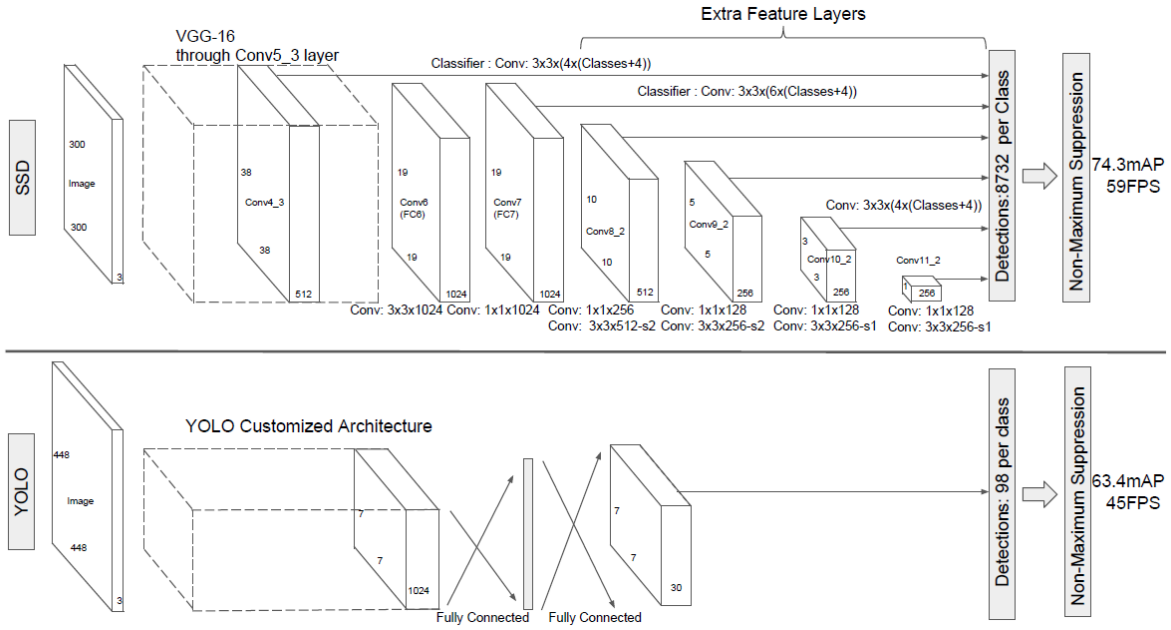


FIGURE 2.26 – Comparaison des architectures SSD et YOLO. Image de Liu *et al.* (2016).

L'architecture SSD s'adapte à tout type d'extracteur de caractéristiques, contrairement à YOLO dont le réseau de base lui est propre. Initialement, c'est VGG-16 qui est utilisé par SSD, avant d'être remplacé par ResNet, et plus récemment par les différentes itérations des MobileNets. Deux types de couches complètent le réseau de base pour former la structure auxiliaire. Tout d'abord, des couches de convolution sont ajoutées à la suite du réseau de base, de tailles de plus en plus réduites pour diminuer progressivement la définition spatiale des couches de caractéristiques. Elles sont appelées « couches *extra* », et ce sont elles qui permettent de détecter des objets de différentes tailles et donnent son aspect « multi-résolution » à l'architecture. Ensuite, le réseau comprend des branches de prédiction : toutes les couches *extra*, ainsi que certaines des couches de caractéristiques du réseau de base, sont utilisées pour effectuer des prédictions grâce à une série de convolutions. Pour chacune de ces couches de caractéristiques, un premier noyau de taille 3×3 produit le score pour toutes les classes, et un second régresse les coordonnées de la boîte englobante par rapport à la boîte par défaut. Dans le premier cas, la sortie est de taille $4 \times b$, avec b le nombre de boîtes par défaut pour chaque position sur les cartes de caractéristiques, et 4 pour les *offsets* de coordonnées (coordonnées x et y du centre, largeur et hauteur). Dans le second cas, la sortie est de taille $k \times b$, avec k le nombre de classes (incluant l'arrière-plan comme classe d'indice 0). Pour une couche de caractéristiques de taille $D_F \times D_F$, cela donne un total de $(k+4) \cdot b \cdot D_F \cdot D_F$ prédictions par couche de caractéristiques. Le nombre de prédictions en sortie est fixe, puisqu'il correspond au nombre de boîtes prédéfinies, et bien supérieur à celui de YOLO : 8732 prédictions pour SSD, contre 98 pour YOLO, sur le jeu de données Pascal VOC et ses 80 classes. Une étape de « suppression des non maxima » (NMS) est ensuite nécessaire pour éliminer les détections multiples d'un objet

dans plusieurs boîtes. SSD ne comprend aucune couche complètement connectée.

Lors de l'entraînement, il est nécessaire d'associer les objets de la vérité-terrain avec les boîtes par défaut. Le choix des auteurs est ici d'associer chaque objet à plusieurs boîtes par défaut : celle de plus grande superposition (déterminé par l'IoU), ainsi que toutes celles dont l'indice de superposition est supérieur à 50%. Cela facilite l'entraînement en permettant à plusieurs boîtes très proches d'être considérées comme positives, c'est-à-dire associées à un objet et non à l'arrière-plan. Les boîtes non associées à un objet de la vérité terrain correspondent à la classe « arrière-plan », et sont considérées négatives. Toutes les boîtes par défaut négatives ne sont pas gardées pour l'entraînement, car elles sont en nombre bien supérieur aux boîtes positives. Elles sont triées par indice de confiance, et les boîtes par défaut d'indices de confiance les plus élevés sont gardées, de façon à garder au maximum trois exemples négatifs pour un positif. Un total de N boîtes positives est gardé pour chaque image d'entraînement ; l'ensemble des boîtes positives est noté Pos, et l'ensemble des boîtes négatives gardées est noté Neg. On définit également m_{ij}^k la variable indiquant si la boîte par défaut i est associée à l'objet de la vérité-terrain d'indice j ($m_{ij} = 1$), ou non ($m_{ij} = 0$), pour la catégorie k.

SSD est optimisé avec une fonction de perte inspirée de l'objectif MultiBox (Erhan *et al.*, 2014), étendu à plusieurs classes. La fonction de perte est une somme pondérée d'un objectif de localisation et d'un objectif de classification, tout comme pour YOLO. L'objectif de localisation est une fonction de coût L1 lisse (Ross Girshick, 2015) calculée entre les coordonnées de la boîte prédite et un objet de la vérité-terrain, et l'objectif de classification est une fonction *softmax* sur les indices de confiance de toutes les classes. Formellement, les coordonnées d'une boîte englobante de la vérité terrain sont définies de la façon suivante, en reprenant les même notations que sur la Figure 2.25 :

$$b_x = \hat{t}_x p_w + c_x \quad b_y = \hat{t}_y p_h + c_y \quad b_w = p_w e^{\hat{t}_w} \quad b_h = p_h e^{\hat{t}_h} \tag{2.10}$$

Pour rappel, $\hat{t}_*$ est un vecteur de coordonnées d'une boîte englobante de la vérité-terrain, et on note t_* un vecteur de prédiction du réseau SSD pour les coordonnées d'une boîte englobante. Il est cependant à noter qu'ici, c_x et c_y correspondent aux coordonnées du centre de la boîte englobante par défaut, et non du coin supérieur gauche de la région associée à la boîte par défaut, comme c'est le cas pour YOLOv3.
La fonction de perte pour la localisation L_{loc} est définie comme suit, avec i l'indice de boîte par défaut et j l'indice d'une boîte de la vérité terrain :

$$L_{loc}(m, t_*, \hat{t}_*) = \sum_{i \in Pos}^{N} \sum_{* \in x,y,w,h} m_{ij}^k \text{smooth}_{L1}(t_{i,*} - \hat{t}_{j,*}) \tag{2.11}$$

En notant C^k l'indice de confiance prédit pour la classe k (la classe 0 correspondant à l'arrière-plan), la fonction de perte associée est définie comme :

$$L_{conf}(m, C) = -\sum_{i \in Pos}^{N} m_{ij}^k log(C_i^k) - \sum_{i \in Neg} log(\hat{C}_i^0) \quad \text{avec} \quad \hat{C}_i^k = \frac{e^{c_i^k}}{\sum_k e^{c_i^k}} \tag{2.12}$$

La fonction de perte finale correspond à l'équation suivante, dans laquelle le coefficient α est fixé à 1 expérimentalement :

$$L(m, c, t_*, \hat{t}_*) = \frac{1}{N}(L_{conf}(m, c) + \alpha L_{loc}(m, t_*, \hat{t}_*)) \tag{2.13}$$

SSD prédit les objets à une vitesse de 59 images par seconde, pour des images d'entrée de taille 300 × 300 pixels, avec VGG-16 comme extracteur de caractéristiques. Dans la publication originale (Liu *et al.*, 2016), les auteurs indiquent que la majorité du temps d'inférence provient du réseau de base, suggérant qu'un extracteur de caractéristiques différent est une direction à étudier pour un réseau SSD plus rapide.

SSD Lite

Alors que dans Sandler *et al.* (2018), les auteurs remplacent le réseau de base VGG-16 de SSD par MobileNetV1 et MobileNetV2, ils modifient également les couches *extra* afin de proposer une architecture moins gourmande en mémoire et plus rapide, nommée SSD Lite. Dans cette variante de SSD, les convolutions standards sont remplacées par des convolutions séparables dans les couches de prédiction, ce qui est cohérent avec l'utilisation des réseaux MobileNets. La différence entre SSD et SSD Lite est mise en valeur dans le Tableau 2.7, avec Mult-Adds correspondant au nombre de séquences d'opérations « multiplication-addition » ; c'est une métrique qui permet une comparaison équitable des réseaux de neurones car elle ne dépend pas de la plateforme sur laquelle ils sont exécutés.

TABLEAU 2.7 – Comparaison des nombres de paramètres et d'opérations (Multiplications/Additions) entre SSD et SSD Lite, avec MobileNetV2 comme réseau de base, pour détecter 80 classes. Tableau tiré de Sandler *et al.* (2018).

	Paramètres (en millions)	**Mult-Adds (en milliards)**
SSD	14,8	1,25
SSD Lite	2,1	0,35

Enfin, SSD Lite est l'architecture adoptée dans Howard *et al.* (2019) pour démontrer les avantages à utiliser MobileNetV3 pour la tâche de détection d'objets. Les auteurs y décrivent comment ils intègrent MobileNetV3 comme extracteur de caractéristiques de SSD Lite. Cette architecture est détaillée sur la Figure 2.27.

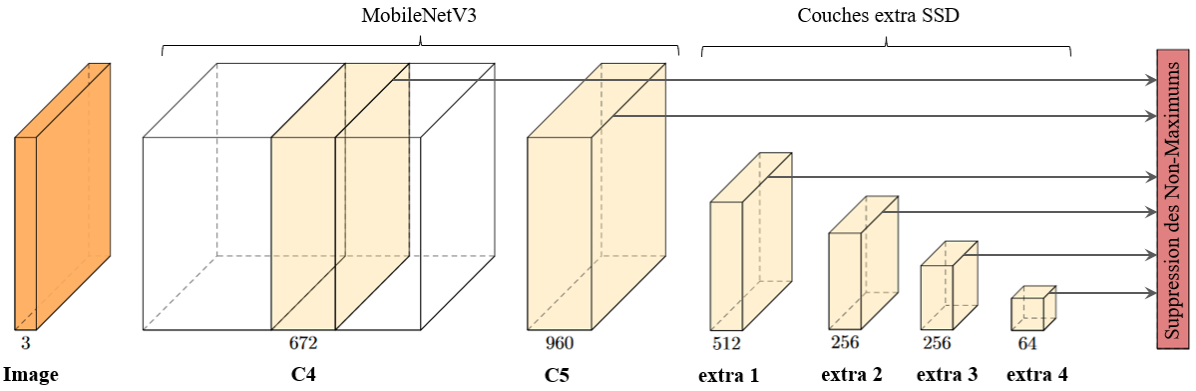


FIGURE 2.27 – Architecture du détecteur d'objets SSD avec MobileNetV3 comme extracteur de caractéristiques. Les dimensions indiquées correspondent à MobileNetV3 *Large*.

Cette étude des réseaux de classification et de détection d'objets de l'état de l'art nous a permis d'orienter nos choix d'architecture. En particulier, l'architecture Faster R-CNN,

qui obtient la meilleure performance sur les différents *benchmarks*, a une durée d'inférence trop importante pour notre application en temps réel. Au contraire, les détecteurs *single-stage* YOLOv3 et SSD correspondent à nos attentes en termes de contraintes de mémoire et de vitesse. SSD présente en outre l'avantage de s'adapter à n'importe quel extracteur de caractéristiques, ce qui permet d'ajuster sa taille et performance en changeant d'architecture. Farasin *et al.* (2020) illustrent bien les différences entre SSD et Faster R-CNN avec le casque de RA HoloLens de Microsoft : en utilisant un serveur de calcul à distance, plus puissant que l'ordinateur embarqué, le délai de transmission rend la détection impossible à effectuer pour chaque image. Ces résultats nous encouragent à utiliser uniquement les ressources de calcul locales du système de RA, sans avoir recours à une unité de calcul externe, ce qui implique un réseau de très petite taille.

2.4 Détection d'objets à partir de modèles 3D

Le capteur le plus simple et disponible sur de nombreux appareils pour acquérir des données étant une caméra, la plupart des réseaux de neurones artificiels de détection d'objets sont appliqués à des images. Cependant, des images ne sont pas toujours disponibles afin d'entraîner le réseau, ou bien ne comportent pas autant d'informations que d'autres modalités, ce qui demande d'intégrer des données différentes. En particulier, la démocratisation de la conception 3D a encouragé l'utilisation de modèles 3D numériques d'objets pour l'analyse d'environnement.

Dans cette partie, nous présentons différentes méthodes de la littérature ayant été proposées pour utiliser des modèles 3D, en commençant par les réseaux de neurones prenant en entrée un volume au lieu d'une image. Ensuite, nous abordons la détection d'objets en trois dimensions et l'estimation de pose en six dimensions. Enfin, nous détaillons l'utilisation d'images RGB-D pour la détection d'objets, et l'application des principes d'augmentation d'images à la modalité de la profondeur.

2.4.1 Les réseaux à convolutions appliqués aux volumes

Alors que les noyaux de convolution classiques s'appliquent communément à des images à 3 canaux (rouge, vert, bleu), un autre type de réseaux de neurones s'est développé avec le temps, appelés réseaux à convolutions 3D (ou 3D CNN). Une convolution 3D considère un noyau de convolution dans trois directions (hauteur, largeur et profondeur) au lieu de deux (hauteur et largeur). Ainsi, le résultat de l'application d'un noyau de convolution 3D à un volume sera un nouveau volume, au lieu d'une surface 2D pour une convolution classique, comme représenté sur la Figure 2.28. Ce type de convolution est appliqué aux images dont la profondeur est supérieure à 3 canaux, comme certaines modalités d'images médicales, des séquences d'images RGB pour l'analyse de vidéos, ou encore à des formes et scènes 3D. C'est ce dernier sujet que nous couvrons dans cette partie. Plusieurs architectures de 3D CNN ont ainsi été proposées pour apprendre des caractéristiques pour les formes 3D : VoxNet (Maturana et Scherer, 2015), 3D ShapeNets (Wu *et al.*, 2015), Octnet (Riegler *et al.*, 2017), CubeNet (Worrall et Brostow, 2018), etc. Leur principal désavantage est qu'ils augmentent fortement le nombre d'opérations effectuées, ce qui limite leur utilisation dans des domaines où l'efficacité est primordiale.

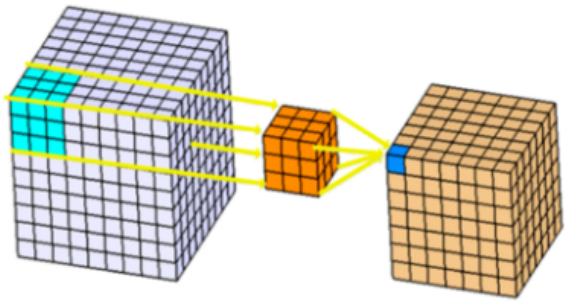


FIGURE 2.28 – Illustration d'une convolution 3D [22] : le noyau est appliqué à une localisation, puis déplacé à la fois dans la hauteur, largeur et profondeur afin de former un nouveau volume.

Dans les travaux portant sur les volumes et formes 3D, les données initiales peuvent être sous forme de nuages de points, ou peuvent être des *voxels*, l'équivalent des pixels en 3 dimensions. Des modèles 3D peuvent ainsi être directement alignés dans la scène (Avetisyan *et al.*, 2019). En ce qui concerne les nuages de points, plusieurs approches ont été proposées, afin de prendre en compte la dimension éparse des données notamment. Le réseau PointNet (Charles *et al.*, 2017) est l'un des plus populaires pour manipuler des nuages de points ; il sert d'extracteur de caractéristiques pour être ensuite utilisé dans différentes applications comme la classification d'objets, la segmentation de parties d'objets, ou la segmentation de scène. Il prend en entrée une séquence de points repérés dans l'espace 3D, et les modifie avec une succession de petits réseaux MLP afin d'obtenir un vecteur de caractéristiques global, et finalement une prédiction de classe. Ce vecteur global peut également être utilisé par un réseau annexe pour produire une segmentation du nuage de points. En parallèle, une autre approche est proposée par Engelcke *et al.* (2017) qui utilise des convolutions éparses appliquées directement au nuage de points, afin d'identifier et localiser des piétons, cyclistes et voitures dans une scène urbaine, ce qui nécessite environ 1 seconde par image (avec uniquement un CPU pour effectuer les calculs). PointPillars (Lang *et al.*, 2019) détecte les mêmes classes en s'appuyant sur les PointNets : l'encodeur se compose de plusieurs PointNets organisés sous forme de colonnes, afin de produire des caractéristiques utilisées ensuite par le détecteur d'objets SSD, à 60 images par secondes. Dans Simon *et al.* (2019), c'est le réseau YOLOv2 qui est modifié pour prendre en entrée des nuages de points afin d'y détecter des objets en 3 dimensions. Enfin, l'architecture GQ-CNN (Mahler *et al.*, 2017) utilise des convolutions classiques appliquées à une carte de profondeur, en parallèle d'une couche complètement connectée appliquée au nuage de points, avant de fusionner les caractéristiques issues des deux modalités. La sortie est une estimation de la réussite d'un robot à attraper l'objet avec une certaine orientation. Entraîné avec pas moins de 6 millions de nuages de points synthétiques, ce modèle y parvient en moins d'une seconde pour des objets connus.

L'inconvénient des nuages de points est qu'ils ont besoin d'un très grand nombre de points afin de décrire précisément un objet, et ils nécessitent souvent un découpage de la scène afin de classifier chaque objet. La détection est particulièrement difficile lorsque plusieurs objets très similaires doivent être identifiés. Ainsi, l'autre approche proposée dans la littérature est la représentation des objets 3D par éléments de volume, les voxels.

22. `https://www.kaggle.com/code/shivamb/3d-convolutions-understanding-use-case/notebook`

L'espace est communément découpé en une grille de voxels en trois dimensions, qui peut être construite à partir d'un nuage de points ou d'un modèle 3D. Chaque voxel est associé à une étiquette, indiquant si celui-ci est « vide », « inconnu » ou « occupé » par un objet, et éventuellement la catégorie de cet objet. Le réseau de neurones 3D ShapeNets (Wu *et al.*, 2015) apprend par exemple une représentation 3D des objets grâce à des modèles 3D représentés par une distribution de probabilités de variables binaires réparties sur une grille de voxels. Assez similaire dans la représentation des formes 3D, VoxNet (Maturana et Scherer, 2015) utilise une carte de profondeur convertie en grille d'occupation volumétrique. Une autre représentation structurée communément adoptée pour les *voxels* est l'*octree*, un arbre représentant un espace 3D en subdivisant de façon récursive un cube en huit plus petits cubes, appelés « octants ». Un octree est une représentation compacte et efficace de l'information 3D, et peut être facilement fournie à un réseau de neurones. Les réseaux O-CNN (Wang *et al.*, 2017), ainsi que son successeur AO-CNN (Wang *et al.*, 2019), ou encore OctNet (Riegler *et al.*, 2017), utilisent des modèles 3D sous forme d'octree afin d'apprendre des caractéristiques liées aux formes géométriques, et produire un vecteur de caractéristiques. Ils sont ensuite utilisés dans différentes applications : classification d'objets, récupération des objets les plus similaires dans un jeu de données, segmentation des différents éléments constituant le volume, etc. Les octree peuvent également être générés par des réseaux de neurones pour produire des volumes à partir de caractéristiques définies (Tatarchenko *et al.*, 2017). Bien que de plus en plus efficaces et rapides, ces méthodes basées sur la décomposition de l'espace en voxels demandent une quantité importante de traitements entre l'acquisition d'une scène et son utilisation en entrée d'un réseau de neurones. Enfin, Hegde et Zadeh (2016) combinent une représentation basée sur les pixels et une seconde basée sur les voxels, pour une meilleure reconnaissance des objets 3D.

2.4.2 Détection d'objets 3D et estimation de pose 6D

Une nouvelle tâche a émergé ces dernières années, mêlant images 2D et information 3D : la détection 3D d'objets (ou de personnes) dans une scène 2D. Cette tâche consiste à localiser un objet dans une image par une boîte englobante en trois dimensions, au lieu d'un simple rectangle. Le *benchmarks* PASCAL3D+ [23] a été proposé pour cela par Xiang *et al.* (2014) ; c'est une extension du jeu de données Pascal VOC avec des annotations 3D pour 12 catégories, obtenues en alignant manuellement des modèles 3D aux objets dans les images.

Si l'annotation d'images réelles est complexe, l'utilisation de données d'entraînement synthétiques est particulièrement efficace pour cette application ; en effet, la position exacte de l'objet étant connue, il est facile d'associer automatiquement une boîte englobante 3D aux objets d'une image, au lieu d'une simple boîte englobante 2D. Pour aller plus loin, c'est même souvent la position précise de l'objet qui est utilisée, c'est-à-dire ses trois coordonnées de position ainsi que ses trois coordonnées de rotation, par rapport à une position « de repos » du modèle 3D. Cette tâche est appelée « estimation de pose », la « pose » faisant référence à la position et à l'angle de rotation d'un objet. Si les images annotées sont simples à obtenir, les méthodes d'estimation de pose s'appliquent généralement à une petite fenêtre dans laquelle se trouve un objet de classe connue ; cela demande donc une première étape de détection de l'objet dans l'image.

23. `https://cvgl.stanford.edu/projects/pascal3d.html`

Le *challenge* BOP (*Benchmark for 6D Object Pose Estimation*[24]) est une référence dans la littérature, puisqu'il propose plusieurs jeux de données comportant des objets (dont les scans ou modèles 3D sont disponibles), des images réelles annotées, ainsi que des images synthétiques également annotées. Ces images contiennent une composante RGB, et une composante de profondeur sous la forme d'une carte de profondeur. Si pendant longtemps, les meilleures performances d'estimation de pose étaient obtenues avec seulement l'une des deux modalités (la couleur ou la profondeur) comme avec le réseau SSD-6D (Kehl *et al.*, 2017), les derniers résultats encouragent l'utilisation des deux, de façon simultanée ou successive. Par exemple, la méthode de Wohlhart et Lepetit (2015) consiste à extraire des caractéristiques avec un CNN à partir de l'image RGB-D afin d'obtenir des descripteurs pour chaque objet et chaque pose 6D. A l'inférence, pour un patch donné contenant un objet, le descripteur est calculé, et son plus proche voisin parmi les descripteurs d'entraînement lui est associé. Le réseau AAE (pour *Augmented Auto-Encoder*) prend en entrée un patch provenant de l'image RGB d'origine, identifié par le réseau SSD comme étant un objet d'intérêt (Sundermeyer *et al.*, 2018). Comme illustré sur la Figure 2.29, AAE est entraîné avec des images synthétiques afin d'extraire l'objet d'une image en éliminant les pixels d'arrière-plan, et en reconstruisant l'objet (notamment s'il est partiellement caché par d'autres objets). Ainsi, un patch provenant d'une image réelle est « nettoyé », afin que l'objet ait une apparence plus proche des images synthétiques. Le code latent obtenu est ensuite associé à celui de l'image du jeu d'entraînement le plus proche, dont la pose est connue. Cette estimation peut encore être affinée en utilisant l'image de profondeur, avec la méthode ICP (*Iterative Closest Point*) (Chen et Medioni, 1992; Zhang, 1994) qui rapproche itérativement le modèle 3D dans la position estimée et la carte de profondeur, jusqu'à ce qu'il se superposent. Ainsi, le simple réseau AAE (sans SSD) ne nécessite que 6ms pour effectuer une prédiction, mais l'affinement de cette prédiction par ICP ajoute plus de 300ms au processus. De plus, un réseau AAE n'est appliqué qu'à un objet du jeu de données, et non à toutes les catégories. L'approche de Kehl *et al.* (2016) est similaire, mais des patches de l'image de profondeur sont utilisés avec ceux de l'image RGB afin d'obtenir les vecteurs de caractéristiques. Les méthodes ultérieures, bien que plus rapides et efficaces, sont basées sur le même principe. Massa *et al.* (2016) produisent des rendus 2D de chaque objet à partir des modèles 3D, et encouragent les caractéristiques extraites d'images réelles à imiter celles de ces rendus 2D synthétiques. Ainsi, la vue synthétique la plus proche peut être associée à l'image réelle durant l'inférence.

D'autres travaux proposent des approches différentes pour l'appariement d'images et de modèles 3D. Avec Pix2Pose (Park *et al.*, 2019), une méthode populaire pour estimer la pose d'un objet à partir d'une simple image, chaque modèle 3D est converti en un « modèle à coordonnées colorées » (Figure 2.30). Ainsi, la prédiction se fait par mise en correspondance des pixels colorés entre l'image prédite et le modèle coloré. A nouveau, cette méthode s'applique à des patches comprenant un seul objet, et peut être améliorée par ICP moyennant un temps de calcul plus important. Pavlakos *et al.* (2017) détectent d'abord les objets en 2D avec Faster R-CNN, puis prédisent des points d'intérêt pour chacun des objets, avant de faire correspondre ces points d'intérêts à ceux définis pour chaque modèle 3D. Enfin, lorsque seules des images RGB sont disponibles à l'inférence, et des images RGB-D non annotées à l'entraînement, Rad *et al.* (2019) proposent d'utiliser des images de profondeur synthétiques (automatiquement annotées) pour entraîner un

24. `https://bop.felk.cvut.cz/home/`

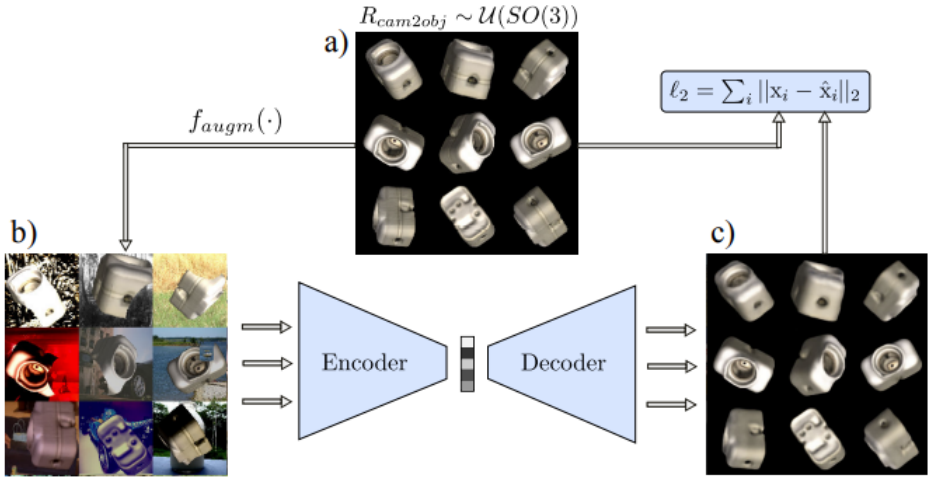


FIGURE 2.29 – Illustration de la phase d'entraînement du réseau AAE (Sundermeyer *et al.*, 2018). a) correspond aux données synthétiques d'entraînement, b) aux mêmes données avec un fond choisi aléatoirement, et c) à la sortie du réseau, qui élimine le fond.

estimateur de pose, et les paires d'images RGB-D réelles pour apprendre à l'extracteur de caractéristiques RGB à imiter l'extracteur de caractéristiques de profondeur. Ces travaux s'apparentent aux méthodes d'adaptation de domaine, un vaste sujet que nous n'aborderons pas ici, par manque de place plus que d'intérêt.

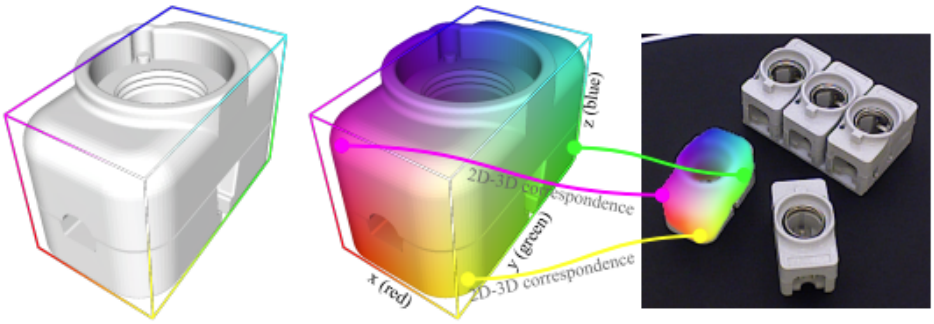


FIGURE 2.30 – Illustration de la méthode Pix2Pose (Park *et al.*, 2019). De gauche à droite : modèle 3D non texturé, modèle à coordonnées colorées, prédiction de Pix2Pose.

Le vainqueur du *challenge* BOP 2020 est CosyPose (Labbé *et al.*, 2020), un estimateur de pose qui utilise les prédictions de détection 2D d'un réseau Mask R-CNN dans plusieurs images de la même scène, afin de proposer des « candidats ». Ces candidats sont des modèles 3D associés à une pose initiale, dont certains sont validés avant d'affiner leurs positions par reconstruction de la scène. Si la performance est la meilleure pour ce *benchmark*, le temps nécessaire pour obtenir les prédictions est cependant trop important pour une utilisation en temps réel et dans une application embarquée (13 secondes pour une image et avec ICP, 449ms sans ICP). Quelques travaux ont tenté de proposer un réseau de neurones prédisant la pose des objets en une seule étape, avec (Kehl *et al.*, 2017; Rad et Lepetit, 2017), ou sans (Tekin *et al.*, 2018) affinement ultérieur. Si cela améliore fortement la vitesse d'inférence, la performance est plus basse que les méthodes plus lentes, de la même façon que les détecteurs d'objets *single stage* sont plus rapides mais moins

performants que les réseaux *double stage*.

Au-delà de l'analyse d'images, les réseaux de neurones ont peu à peu été utilisés pour remplacer des méthodes génératives, que ce soit pour générer des images ou des modèles 3D. Ainsi, l'approche *Render-and-Compare* consiste à générer un modèle 3D à partir d'une image 2D (ou choisir le modèle 3D le plus proche dans un ensemble connu), estimer sa pose, puis générer un rendu 2D de ce modèle afin de le comparer à l'image d'origine (Kundu *et al.*, 2018). Dans la même idée, Gupta *et al.* (2015) effectuent d'abord une segmentation de chaque instance, avant d'ajuster des modèles 3D connus aux masques obtenus en minimisant l'erreur de projection.

En résumé, bien que les méthodes d'estimation de pose permettent de localiser très précisément un objet dans une image en ne connaissant que son modèle 3D, elles dépendent presque toutes de la qualité de la détection 2D préalable, et nécessitent ensuite pour la plupart un estimateur de pose différent pour chaque catégorie. Bien que des méthodes plus rapides existent, elles ont une performance plus basse. Parmi les jeux de données utilisés, nous avons également observé que les objets sont toujours de petite taille et visibles en entier dans les images, contrairement aux objets des images égocentriques d'apparence plus complexes. Enfin, l'annotation d'images réelles pour l'évaluation de ces méthodes est complexe, et nécessite une étude de tous les objets considérés afin de gérer correctement la question de la symétrie, qui peut fausser les prédictions.

2.4.3 Détection d'objets 2D dans des images RGB-D

Les modèles 3D et capteurs de profondeur étant facilement disponibles ces dernières années, nous nous sommes intéressés aux approches de la littérature utilisant ces deux modalités en parallèle de l'image couleur. La détection d'objets avec des images RGB-D nous permet également d'aborder le sujet de la fusion multimodale et de l'entraînement avec de multiples tâches. Les travaux mentionnés par la suite sont généralement appliqués à la détection d'objets 2D, mais certains proposés dans le cadre de la segmentation sémantique ou de la reconnaissance d'actions apportent des idées complémentaires sur la façon d'utiliser l'information 3D.

Définissons tout d'abord les différents types de fusion de modalités. On parle de fusion précoce (*early fusion* en anglais) lorsque les différentes modalités sont combinées avant d'être fournies à un algorithme. Dans le cas des images RGB-D, cela correspond à la concaténation des images de couleur et de profondeur, afin d'obtenir une image à 4 canaux (rouge, vert, bleu, profondeur) (Zhao *et al.*, 2020), ou plus si la carte de profondeur est elle-même décomposée selon plusieurs canaux (pseudo-RGB (Eitel *et al.*, 2015), ou HHA (Gupta *et al.*, 2014)). A l'opposé, des prédictions peuvent être produites pour chaque modalité de façon entièrement séparées, avant qu'elles ne soient combinées dans une approche ensembliste (Xu *et al.*, 2017) ou analysées pour choisir la prédiction la plus plausible (Fan *et al.*, 2021). Entre ces deux extrêmes, ce sont les caractéristiques issues des différentes entrées qui sont combinées, à des endroits différents dans un réseau de neurones : juste avant la couche de prédiction pour une fusion tardive (*late fusion*) (Eitel *et al.*, 2015; Cao *et al.*, 2017), ou plus tôt pour une fusion intermédiaire (*mid-level fusion*) (Hoffman *et al.*, 2016; Hou *et al.*, 2016; Ophoff *et al.*, 2019).

Plusieurs travaux ont exploré différentes décompositions d'un réseau de neurones pour la fusion intermédiaire ou tardive (Roitberg *et al.*, 2019; Ophoff *et al.*, 2019). En effet, ces types de fusion impliquent des couches de caractéristiques spécifiques à chaque modalité en début d'architecture, une étape de fusion de ces deux branches, puis un ensemble de

couches communes. La question de la « meilleure » couche pour effectuer la fusion se pose donc. Les expériences menées semblent converger vers une fusion intermédiaire lorsque les modalités en entrée sont de nature similaire comme différents types d'images, et une fusion plus tardive pour des modalités plus différentes telles qu'une image et l'audio associé (Neverova *et al.*, 2016). Plusieurs méthodes d'initialisation et de pré-entraînement ont également été proposées : pré-entraîner chaque branche séparément avec sa modalité, puis assembler le réseau et continuer l'entraînement (Roitberg *et al.*, 2019; Ophoff *et al.*, 2019; Douarre *et al.*, 2020) ; entraîner la branche RGB, puis utiliser ces poids pour initialiser la branche de profondeur avant d'assembler le réseau (Hoffman *et al.*, 2016) ; pré-entraîner les deux branches avec une tâche auto-supervisée (Zhao *et al.*, 2021), etc. De façon générale, elles s'accordent sur l'avantage d'un entraînement par modalité, puis d'un entraînement conjoint.

Alors que les méthodes précédentes ont plutôt une approche *single stage*, ou de « bout-en-bout », il est aussi possible de décomposer les différentes étapes d'extraction des caractéristiques et de détection. Par exemple, Gupta *et al.* (2014) produisent une carte des contours à partir de l'image RGB-D, utilisée pour l'étape de proposition de régions. De même, Hou *et al.* (2016) dérivent différentes caractéristiques des images RGB et de profondeur (comme la carte de contours ou la hauteur par rapport au sol), concatènent les vecteurs de caractéristiques obtenus grâce à différents CNNs, avant de les fournir à un réseau R-CNN suivi d'un SVM pour détecter les objets. Schwarz *et al.* (2018) explorent différentes façons d'utiliser l'image de profondeur pour complémenter l'image couleur : pour extraire des propositions de régions, et pour extraire des caractéristiques.

Une autre architecture a été proposée pour la fusion intermédiaire : l'utilisation d'une troisième branche dédiée à l'extraction de caractéristiques communes aux deux modalités, afin que les deux autres branches se spécialisent sur les caractéristiques spécifiques à chaque modalité. Cette nouvelle branche peut être totalement indépendante des deux autres, ou bien y être liée jusqu'à la prédiction. Par exemple, Kim *et al.* (2018) utilisent deux réseaux SSD dont les caractéristiques sont combinées à plusieurs niveaux, et les prédictions ne sont faites qu'à partir des caractéristiques fusionnées. Leurs modules de fusion GIF font partie des quelques architectures proposées pour apprendre à combiner les modalités, tout comme les blocs *Cyclic Fuse-and-Refine* Zhang *et al.* (2020), ou encore les *cross-stitch units* (Roitberg *et al.*, 2019). Xu *et al.* (2017) proposent d'utiliser trois branches pour extraire des caractéristiques et effectuer des prédictions selon le modèle Fast R-CNN, avant de combiner ces prédictions dans une logique ensembliste. L'approche proposée par Hoffman *et al.* (2016) comprend également trois branches, mais ici la troisième branche sert à « halluciner » l'image de profondeur : elle reçoit une image RGB en entrée, mais est contrainte à imiter les caractéristiques de la branche de profondeur pendant l'entraînement. Cela permet de n'utiliser que l'image RGB à l'inférence, en ne gardant que la branche RGB et la branche « d'hallucination », tout en apprenant des images de profondeur pendant l'entraînement. Une approche différente pour entraîner un réseau avec des images RGB-D lorsque seule l'image RGB est disponible à l'inférence est l'entraînement d'un réseau d'estimation de profondeur à partir de l'image RGB (Cao *et al.*, 2017). Avec un auto-encodeur comme réseau principal, Zhao *et al.* (2021) utilise un encodeur par modalité, puis un décodeur commun intégrant des modules de fusions « inter-modalités » (pour combiner les modalités) et « inter-niveaux » (pour propager l'information sémantique acquise avec une définition spatiale inférieure).

Plutôt que de considérer l'image RGB et la carte de profondeur comme deux sources fournissant le même type d'information, il est possible de les utiliser de façon successive et

complémentaire. Xie *et al.* (2020) obtiennent des masques initiaux à partir de l'image de profondeur, avant de les affiner par opérations morphologiques. L'image RGB est ensuite utilisée avec les masques en entrée d'un second réseau de neurones, pour une segmentation plus précise.

Enfin, les possibilités offertes par l'information 3D disponible ont encouragé le développement de méthodes résolvant plusieurs tâches à la fois, en parallèle de la tâche principale. Ces architecture multi-tâches s'assurent ainsi d'extraire des caractéristiques variées et complémentaires, comme Liang *et al.* (2019) qui estiment le plan correspondant au sol dans l'image de profondeur, prédisent des détections 2D et 3D, et complètent l'image de profondeur éparse obtenue avec un capteur de type LiDAR.

2.4.4 Augmentation et pré-traitement des images de profondeur

Lors de l'utilisation d'images RGB-D, la question de l'augmentation de la carte de profondeur se pose. En effet, comme pour une image RGB, il est nécessaire de faire varier l'apparence des images montrées au réseau afin de le rendre plus robuste aux variations des images de test. Une carte de profondeur correspond à une image dans laquelle chaque pixel encode la distance à l'objet représenté à cette position. En général, ces images de profondeur sont augmentées uniquement avec les opérations géométriques définies pour les images RGB, car modifier les pixels altère l'information de distance. De plus, les images de profondeur présentent souvent des défauts, inhérents au fonctionnement des capteurs de profondeur. Ces défauts se traduisent par un certain nombre de pixels pour lesquels la valeur n'est pas connue ni calculable, en raison de plusieurs phénomènes :

- La carte de profondeur est reconstruite à partir de capteurs stéréoscopiques, puis alignée avec l'image couleur. Tous les points de l'espace visibles dans l'image couleur ne sont donc pas visibles dans la reconstruction de la profondeur de la scène, et cela se traduit par des pixels de valeur nulle sur certains contours d'objets lors de l'alignement des deux modalités.
- Les surfaces réfléchissantes ont la particularité d'être souvent mal reconstruites, car la lumière infra-rouge n'est pas correctement renvoyée vers le capteur.

Il peut alors être nécessaire d'améliorer la qualité de ces images avant de les utiliser, à l'entraînement comme à l'inférence, par un pré-traitement spécifique. En particulier, si le réseau est entraîné sur des images synthétiques, ces défauts rendent les images réelles visuellement très différentes ; cela crée un écart de domaine important qui pénalise la performance qu'il est possible d'obtenir.
De nombreuses solutions existent pour répondre à ce dernier problème, afin de rendre les images de profondeur synthétiques plus réalistes. On peut les diviser en deux grandes familles. D'un côté, les images d'entraînement synthétiques peuvent inclure ces défauts, afin qu'ils soient appris par le réseau de neurones lors de l'entraînement. Pour cela, les capteurs et phénomènes physiques en jeu doivent être modélisés avec soin dans l'outil de génération d'images de profondeur (Gschwandtner *et al.*, 2011; Planche *et al.*, 2017). Cependant, cette modélisation ne correspondra précisément qu'à un seul capteur ou type de capteur, et le modèle entraîné ne sera pas efficace sur des images provenant d'un autre capteur, avec d'autres caractéristiques et défauts. Par exemple, Baruhov et Gilboa (2020) utilisent des images d'un capteur de haute qualité pour améliorer les images d'un capteur bas de gamme, montrant bien la différence entre les deux types d'images. Au contraire, Ranftl *et al.* (2022) utilisent des images de profondeur provenant de multiples jeux de

données afin d'obtenir un modèle qui généralise à tous les capteurs. Il est également possible de simuler les défauts sur les images d'entraînement avec une approche par transfert de style (Sweeney *et al.*, 2019), ou plus simplement en copiant les pixels bruités provenant d'images réelles (Eitel *et al.*, 2015). En utilisant un jeu de données cible incluant différents types de capteurs et de défauts, les images synthétiques ne présentant aucun défaut se voient appliquer un style plus réaliste. Dans la pratique cependant, les travaux se limitent à un jeu de données, donc à un capteur, spécifique.

Une autre voie consiste à améliorer les images réelles avant de les donner au réseau de neurones. Pour une application en temps réel, ce traitement doit pouvoir se faire à l'inférence dans un temps très réduit. Parmi les différentes pistes d'amélioration, la complétion des pixels manquants dans les images est un enjeu important ; il permet de compléter la géométrie des objets, une caractéristique importante pour les identifier. Nous présentons dans les paragraphes suivants différentes méthodes permettant de compléter les pixels manquants dans des images de profondeur : l'extension des travaux de Levin *et al.* (2004) sur la colorisation d'images, l'utilisation de réseaux de neurones à convolutions, et une méthode de traitement d'images classique.

Colorisation par optimisation

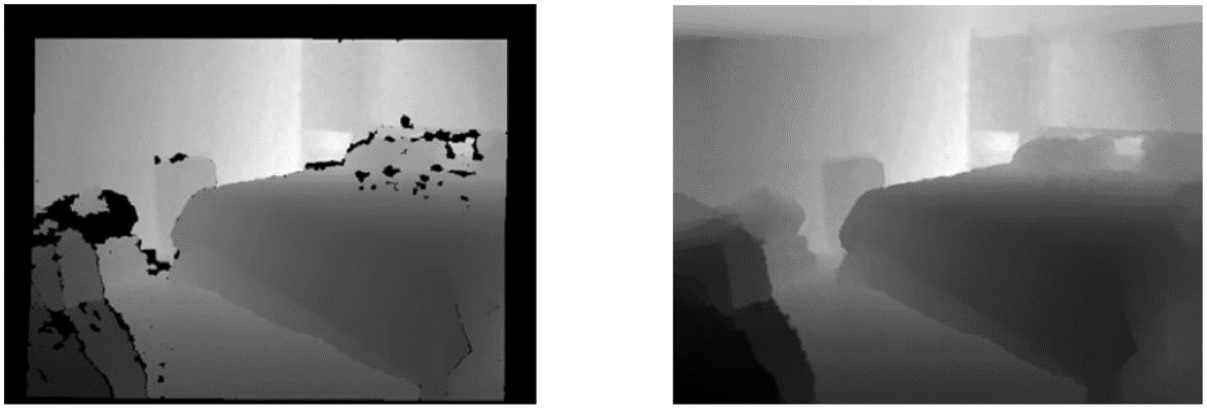
La méthode proposée par Levin *et al.* (2004) est une méthode de référence pour la colorisation d'une image en noir et blanc à partir d'une trace de l'utilisateur. De façon simplifiée, cet algorithme suppose que des pixels proches spatialement (et temporellement dans le cas d'une vidéo) et d'intensités similaires devraient avoir des couleurs similaires. Ainsi, il propage la couleur de la trace utilisateur aux pixels voisins de même niveau de gris, ce qui permet de coloriser toute l'image de proche en proche, de façon itérative. L'algorithme fonctionne par la minimisation de la différence entre la couleur d'un pixel et la moyenne pondérée de ses pixels voisins, d'où son nom de colorisation par optimisation. Cette méthode a été appliquée à la complétion d'images de profondeurs, avec ou sans image RGB associée. Elle est notamment utilisée pour compléter les images du jeu de données public NYU Depth (Silberman *et al.*, 2012), dont un exemple est représenté sur la Figure 2.31. Les pixels vides de l'image de profondeur sont les valeurs à prédire, par élargissement des régions de pixels connus, en utilisant ou non l'image RGB dont toutes les valeurs sont connues. L'inconvénient de cette méthode est son temps d'exécution : plusieurs minutes par image lorsque le calcul est effectué par un ordinateur de bureau classique. Par la suite, différentes méthodes d'optimisation ont été utilisées, y compris en association avec de l'apprentissage profond (Zhang et Funkhouser, 2018).

Complétion de la profondeur par apprentissage profond

Quelques méthodes d'apprentissage profond ont récemment été proposées pour compléter les pixels manquants d'une image de profondeur, et de façon plus générale améliorer sa qualité (Shivakumar *et al.*, 2019; Tang *et al.*, 2021; Hu *et al.*, 2021). Néanmoins, l'utilisation d'un réseau de neurones à convolutions rend l'opération lente, et les données utilisées pour l'entraîner doivent permettre de généraliser suffisamment aux différents objets et capteurs. Les résultats sont souvent décevants lorsque le modèle est entraîné avec des images qui ne sont pas suffisamment représentatives d'images réelles.

Méthode par traitement d'images classique

Les auteurs de Ku *et al.* (2018) prennent le contre-pied de l'apprentissage profond en revenant à un algorithme de traitement d'images classique. Ils proposent de compléter



(a) Profondeur brute.

(b) Profondeur complétée par optimisation avec la méthode de Levin *et al.* (2004).

FIGURE 2.31 – Exemple d'une carte de profondeur du jeu de données NYU Depth (Silberman *et al.*, 2012).

les pixels manquants d'une image de profondeur issue d'un capteur de type LiDAR avec des opérations morphologiques successives. L'algorithme proposé commence par inverser les images de profondeur, pour les pixels connus, afin que les objets les plus proches correspondent aux valeurs les plus grandes. Ensuite, ils appliquent successivement des opérations morphologiques puis une extension des valeurs jusqu'au haut de l'image. Ils terminent par l'application de flou médian et Gaussien. Enfin, l'image est à nouveau inversée. Pour plus de détails, le lecteur peut se référer à la section 5.3.2.

2.5 Point de vue égocentrique

L'analyse d'images égocentriques est un sujet très récent, probablement dû à la difficulté d'acquisition jusqu'à encore très récemment. En effet, les images classiques (« exocentrique ») sont faciles à obtenir avec un appareil photo ou un téléphone portable. Au contraire, les images égocentriques nécessitent un support spécifique afin d'acquérir des images, telles qu'un support pour placer la caméra sur la poitrine ou la tête, ou bien un casque ou des lunettes équipés d'une caméra. Ainsi, la plupart des méthodes de traitement d'images ne s'appliquent pas directement aux images égocentriques, et les algorithmes d'apprentissage profond généralisent difficilement à ce point de vue. Cependant, le développement des supports de caméras, notamment pour les sportifs souhaitant partager leur activité avec une vidéo à la première personne, ainsi que l'engouement pour les réalités augmentée et virtuelle, ont encouragé la recherche à explorer cette voie. Le nombre de publications scientifiques sur le sujet, en constante augmentation depuis 2013, peut en attester (Figure 2.32). Nous résumons dans cette partie les travaux de la littérature liés aux images égocentriques, notamment les jeux de données disponibles et les tâches auxquelles les différentes méthodes proposées s'attaquent à résoudre.

25. `https://app.dimensions.ai/discover/publication`

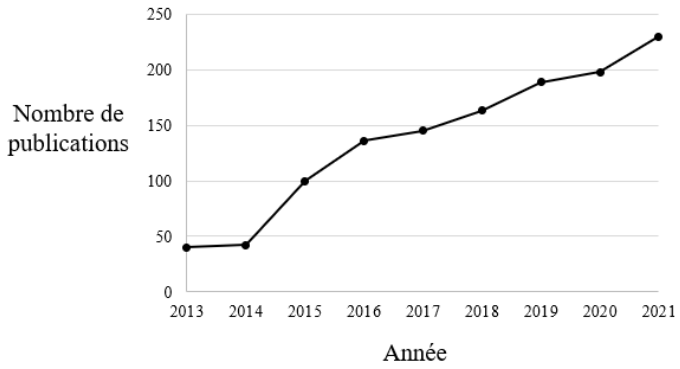


FIGURE 2.32 – Evolution du nombre de publications scientifiques comportant le terme « egocentric » dans le titre ou dans le résumé, dans le domaine de l'intelligence artificielle et du traitement d'images. Données consultées le 07/04/2022 [25].

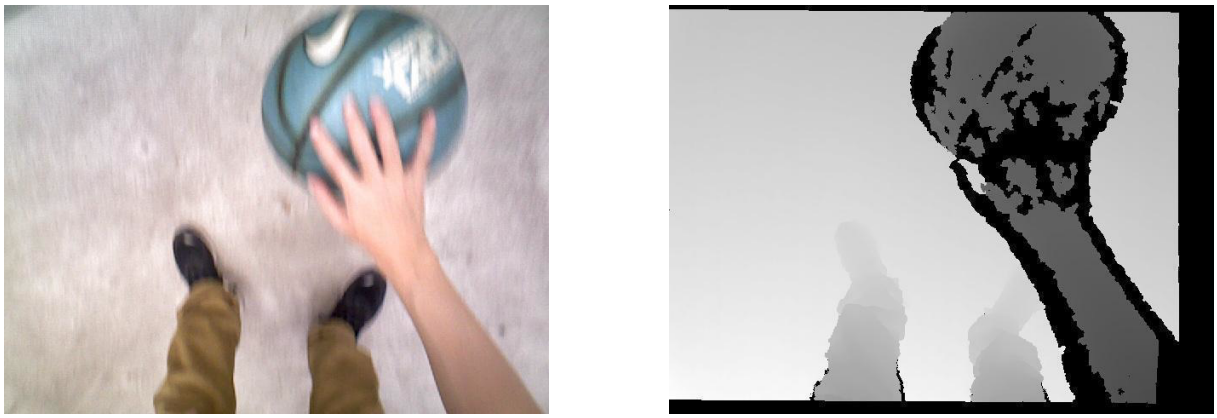
2.5.1 Jeux de données égocentriques

Alors que les jeux de données tels qu'ImageNet ont vocation à être génériques afin de reconnaître un maximum d'éléments de la vie quotidienne, les jeux de données égocentriques répondent plutôt à des applications spécifiques. Il s'agit souvent de vidéos, RGB ou RGB-D, qui ne disposent pas toujours d'annotations pour la détection d'objets. Nous passons en revue ici les jeux de données principaux de la littérature ; ils sont détaillés dans le Tableau 2.8.

Les premiers jeux de données publics ont été proposés dans le cadre de l'analyse d'actions et d'activités : GTEA Dataset [26] (Fathi *et al.*, 2011b), First-Person Social Intractions Dataset [27] (Fathi *et al.*, 2012), HUJI EgoSeg Dataset [28] (Poleg *et al.*, 2014), EDUB-Obj [29] (Bolaños et Radeva, 2015). Pour GTEA, les mains ainsi que l'objet manipulé sont segmentés par un algorithme. Pour EDUB-Obj, 21 classes d'objets sont également segmentées. Plus récemment, des jeux de données aux applications très spécifiques ont été publiés : EgoCart [30] (Spera *et al.*, 2018) pour la localisation dans un supermarché, EPIC-KITCHENS [31] (Damen *et al.*, 2021) pour la reconnaissance d'actions et d'objets en lien avec la cuisine. Enfin, durant cette thèse, le sujet de l'analyse d'images égocentriques a beaucoup évolué, ce qui a mené à la publication du jeux de données Ego4D [32] (Grauman *et al.*, 2021), comprenant plus de 3670 heures de vidéos égocentriques pour l'analyse d'activités quotidiennes. Ce projet rassemble notamment les chercheurs les plus actifs dans le domaine de l'analyse d'images égocentriques, parmi les pas moins de 85 auteurs.

EgoHands [33] (Bambach *et al.*, 2015) contient 48 vidéos égocentriques RGB montrant une personne manipulant des objets, en interaction avec une autre personne. Pour chaque vidéo, 100 images sont annotés avec la segmentation des mains de l'utilisateur, mais les objets ne sont pas annotés. THU-READ [34] (Tang *et al.*, 2017) propose des images RGB-D

26. http://ai.stanford.edu/~alireza/GTEA/
27. http://ai.stanford.edu/~alireza/Disney/
28. https://www.vision.huji.ac.il/egoseg/videos/dataset.html
29. http://www.ub.edu/cvub/edub-obj/
30. https://iplab.dmi.unict.it/EgocentricShoppingCartLocalization/
31. https://epic-kitchens.github.io/2022
32. https://ego4d-data.org/
33. http://vision.soic.indiana.edu/projects/egohands/
34. http://ivg.au.tsinghua.edu.cn/dataset/THU_READ.php



(a) THU-READ (Tang *et al.*, 2017).

(b) MECCANO (Ragusa *et al.*, 2021).

FIGURE 2.33 – Exemples d'images extraites de jeux de données égocentriques.

égocentriques présentant 40 actions manuelles, éventuellement en interaction avec un objet (Figure 2.33a). Cependant, les seules annotations correspondent à la catégorie d'action pour chaque séquence vidéo.

Enfin, un jeu de données égocentrique plus proche de notre application a également été publié durant cette thèse : le MECCANO Dataset [35] (Ragusa *et al.*, 2021), dont le but est la reconnaissance d'objets et d'actions pour la fabrication d'un produit industriel, en suivant un guide d'assemblage. Comme le montrent les images de la Figure 2.33b, on y retrouve les mêmes problématiques que nous avons rencontré dans cette thèse en raison du point de vue égocentrique : le flou de mouvement, les variations de luminosité, les objets sortant du champ de la caméra et partiellement occultés par les mains de l'utilisateur. Bien que l'objet assemblé ici soit un jouet de petite taille, ces travaux rejoignent cette thèse comme l'un des premiers efforts de la communauté pour appliquer les résultats de l'analyse d'images égocentriques à un contexte industriel.

2.5.2 Des tâches associées plus complexes

Comme en témoignent les jeux de données égocentriques publiés, les tâches liées à ces images sont plus complexes que de la simple détection d'objets. Pour commencer, elles sont souvent liées à des séquences d'images ou des vidéos, et non à de simples images isolées, même si l'intervalle entre images successives est très variable. Ces séquences sont souvent utilisées pour la reconnaissance d'activités ou d'actions du quotidien, comme la marche, la lecture ou encore la cuisine (Fathi *et al.*, 2011b,a; Bertasius *et al.*, 2017; Tang *et al.*, 2019).

Ensuite, les objets détectés dans les images égocentriques ne sont en général pas tous les objets, mais seulement ceux en lien avec l'activité en cours ou en interaction avec

35. `https://iplab.dmi.unict.it/MECCANO/`

TABLEAU 2.8 – Détails des jeux de données égocentriques de la littérature.

Jeu de données	**Contenu**	**Modalités**	**Annotations**
GTEA (Fathi *et al.*, 2011b)	31 222 images filmées par 4 personnes montrant 7 activités	RGB	Etiquette de l'objet le plus saillant pour chaque activité
First-Person Social Interaction (Fathi *et al.*, 2012)	42h de vidéos (30fps) filmées par 8 personnes montrant 6 activités	RGB	Segmentation temporelle des activités
HUJI EgoSeg (Poleg *et al.*, 2014)	122 vidéos montrant 14 activités	RGB	Segmentation temporelle des activités
EDUB-Obj (Bolaños et Radeva, 2015)	4912 images filmées par 4 personnes montrant 21 objets	RGB	Segmentation des objets
EgoCart (Spera *et al.*, 2018)	19 531 images	RGB-D	Localisation spatiale dans un supermarché et étiquette de la zone
EPIC-KITCHENS (Damen *et al.*, 2021)	100h de vidéos avec narration, 97 verbes d'actions et 300 noms d'objets	RGB flux optique gyroscope accéléromètre	Segmentation temporelle des activités et segmentation des objets et mains
EgoHands (Bambach *et al.*, 2015)	4800 images	RGB	Segmentation et identification des mains
THU-READ (Tang *et al.*, 2017)	1920 vidéos courtes filmées par 8 personnes montrant 40 actions	RGB-D	Etiquette d'action pour chaque séquence
MECCANO (Ragusa *et al.*, 2021)	20 vidéos filmées par 20 personnes, 12 verbes, 20 objets et 61 actions	RGB	Segmentation temporelle des actions et segmentation des objets
Ego4D (Grauman *et al.*, 2021)	3670h de vidéos	RGB	Annotations diverses pour 16 tâches

l'utilisateur (Bertasius *et al.*, 2017; Tang *et al.*, 2017; Leonardi *et al.*, 2022), voire même le prochain objet actif (Furnari *et al.*, 2017). Cela rappelle la tâche de détection d'objets saillants, (*Salient Object Detection*), dans laquelle les contours de l'objet principal sont délimités par rapport à l'arrière-plan (Borji *et al.*, 2015). Ainsi dans une image, tous les objets ne sont pas des objets d'intérêt et ne sont donc pas annotés. La segmentation des mains est également une tâche récurrente pour les images égocentriques, puisqu'elles apportent un indice important sur les objets en interaction ou les actions de l'utilisateur (Li et Kitani, 2013). De façon générale, les situations considérées correspondent à des activités du quotidien. Alors que seuls quelques travaux très récents s'intéressent aux images égocentriques dans un contexte industriel (Ragusa *et al.*, 2021; Tavakoli *et al.*, 2021; Leonardi *et al.*, 2022), les travaux antérieurs les plus proches sont souvent associés à la manipulation robotique. Si le point de vue est assez similaire, les images acquises par un robot ne présentent pas les mêmes artefacts liés aux mouvement de la tête d'un utilisateur.

Pour terminer, il est à noter que les travaux autour des images égocentriques sont

très souvent à caractère multi-tâche ou multi-modalités. En effet, les vidéos bénéficient de l'intégration de la dimension temporelle (Kazakos *et al.*, 2021), et sont parfois associées à une image de profondeur (Tang *et al.*, 2019). Plusieurs travaux ont également proposé une approche auto-supervisée guidée par la trajectoire de l'utilisateur (*ego-motion*) pour apprendre des caractéristiques représentatives des images (Agrawal *et al.*, 2015; Jayaraman et Grauman, 2015; Yuan et Kitani, 2018; Bozorgtabar *et al.*, 2019; Tsutsui *et al.*, 2021). Enfin, un grand nombre de ces travaux utilisent des images synthétiques afin de générer différentes modalités, différents points de vue, ou les annotations nécessaires.

2.6 Conclusion

Ce chapitre introduit les notions et travaux de la littérature sur lesquels se base cette thèse. Nous nous sommes concentrés sur les méthodes de détection d'objets basées sur de l'apprentissage profond supervisé. En effet, les réseaux de neurones à convolution offrent de nombreuses possibilités lorsque nous avons à disposition des modèles 3D des objets à détecter, c'est pourquoi nous avons exploré différentes pistes afin de les utiliser pour la détection dans des images RGB et RGB-D. En particulier, ils sont capables de généraliser à des situations inconnues et sont invariants à de nombreux paramètres lorsqu'ils sont entraînés correctement : luminosité, occultations, etc.

Alors que les détecteurs d'objets *double stage* sont aujourd'hui très performants sur les jeux de données de référence, ils demandent une grande quantité de calculs qui n'est malheureusement pas compatible avec les applications embarquées et en temps réel. Au contraire, les architectures *single stage* semblent être une alternative intéressante qui permet de respecter les contraintes de mémoire et de temps de calcul des applications embarquées dans des appareils mobiles. Les réseaux de neurones YOLO et SSD en sont des exemples, et le développement d'extracteurs de caractéristiques tels que MobileNet renforce la flexibilité de ces architectures en permettant d'adapter facilement leur taille. Leur capacité d'apprentissage reste cependant limitée, et demande d'être étudiée au cas par cas en fonction des images considérées et des objets à reconnaître.

Alors que nous disposons de modèles CAO des objets à reconnaître, la plupart des méthodes opérant directement sur des objets 3D sont difficilement applicables dans notre contexte. Lors de la mise en situation réelle, seule une caméra sera disponible sur le casque de réalité augmentée, qui peut éventuellement être une caméra RGB-D. Cependant, les caméras du commerce n'ont pas toujours une excellente résolution, et les mouvements permanents de l'utilisateur rendent la reconstruction des images plus complexe. La mise en correspondance entre un modèle CAO et un nuage de points ou une carte de profondeur bruités est une approche très lente, souvent itérative ou par l'application d'une fenêtre glissante, en particulier lorsque nous ne savons pas à l'avance quels objets sont présents dans la scène. Les réseaux à convolution 3D sont particulièrement lents en raison du nombre d'opérations à effectuer, et les réseaux de classification de formes basés sur les voxels demandent d'abord d'extraire l'objet de la scène, sous forme de volume. Ces différentes approches, très prometteuses lorsque le temps de calcul n'est pas limité, sont trop lentes pour notre application.

En ce qui concerne la détection d'objets 3D et l'estimation de pose 6D, les images réelles que nous avons obtenu lors des premiers tests ont montré des symétries et des objets seulement partiellement visible, entre autres artefacts, ce qui augmente grandement la difficulté de la tâche. Nous avons donc choisi de nous concentrer sur l'application des détecteurs d'objets 2D à ces images, plus largement utilisés et maîtrisés. Ces travaux

pourront ensuite être étendus par l'utilisation des mêmes images afin de résoudre une tâche plus complexe, en s'appuyant sur les conclusions de cette thèse.

Les images RGB-D apportent un complément d'information grâce à la carte de profondeur. Il faut cependant l'intégrer de façon réfléchie dans le processus d'apprentissage, afin de limiter les étapes de pré-traitement (pour l'image de profondeur), et adapter l'architecture du détecteur d'objets sans augmenter de trop la taille de l'architecture et le nombre d'opérations. Le premier point est particulièrement important lorsque les cartes de profondeur d'entraînement sont générées de façon synthétique : l'écart à la réalité doit être réduit afin de permettre au réseau de neurones de généraliser aux images réelles. En ce qui concerne l'architecture, les réseaux proposés dans la littérature proposent plusieurs pistes intéressantes concernant la fusion de modalités, même si certains modules de fusion ajoutent une quantité importante d'opérations.

Les images égocentriques bénéficient d'un intérêt croissant les dernières années, comme en témoignent les différents jeux de données publiés récemment. Ils restent cependant très spécifiques et ne proposent pas l'annotation de tous les objets. De plus, ils présentent des objets et situations du quotidien, ce qui diffère du contexte industriel impliquant des objets spécifiques, souvent non texturés. Le traitement d'images égocentriques dans le milieu industriel est encore très peu développé, c'est pourquoi nous nous attachons à soulever les difficultés et proposer des solutions aux problèmes spécifiques qui y sont liés. De plus, nous cherchons une approche qui puisse être appliquée à différents jeux de données industriels, comportant des objets de tailles et formes variées, et qui soit la plus automatisée possible. En effet, la génération d'un jeu de données synthétique ainsi que l'entraînement du réseau de neurones doivent être effectués automatiquement lorsque les modèles 3D sont rendus disponibles par les ingénieurs et designers. Une telle approche pourra ainsi être utilisée dans des contextes non liés à l'industrie, et sans nécessiter d'expert en apprentissage profond. Le sujet de la génération d'un jeu de données synthétiques est abordé de façon détaillée dans le prochain chapitre.

Chapitre 3

Génération de données synthétiques avec des modèles 3D

Dans ce chapitre, nous abordons en détail le sujet des méthodes de génération de jeux de données synthétiques, données utilisées pour l'entraînement de réseaux de neurones pour la détection d'objets spécifiques tels que des objets industriels. Après avoir présenté les différentes approches de l'état de l'art, nous détaillons les deux méthodes que nous avons utilisées pour obtenir des jeux de données d'entraînement composés d'images RGB et RGB-D, respectivement. Tout d'abord, nous décrivons la méthode que nous avons proposée pour créer des images RGB à partir des modèles 3D non texturés du siège de bus RUSPA, en incluant des caractéristiques d'images égocentriques. Ensuite, nous décrivons les images synthétiques RGB-D que nous avons généré à partir d'une scène 3D.

Sommaire

3.1 Notions de base et état de l'art

Le terme d'image synthétique désigne une image générée ou modifiée par ordinateur. Les premiers travaux se sont intéressés aux images générées synthétiquement pour apprendre avec aucune ou très peu d'images réelles (Peng *et al.*, 2015). En effet lorsqu'une classe présente très peu d'exemples, il est possible d'améliorer son taux de reconnaissance en ajoutant aux données originale des images synthétiques correspondant à cette classe. Par la suite, le développement de modèles d'apprentissage profond demandant toujours plus données a rendu extrêmement coûteuse l'action d'acquérir et d'annoter suffisamment de données d'entraînement, ce qui a mené à l'utilisation de jeux de données entièrement synthétiques (Su *et al.*, 2015; Dwibedi *et al.*, 2017; Sarkar *et al.*, 2017; Hinterstoisser *et al.*, 2018; Sundermeyer *et al.*, 2018). Dans ce contexte, l'utilisation d'un jeu de données synthétique, ou artificiel, présente de nombreux avantages. En effet, la génération de données synthétiques permet la création automatique des annotations associées, ce qui évite l'étape d'annotation manuelle. De plus, de nombreux paramètres peuvent être ajustés, afin de générer des images variées et contrôler leur contenu. Enfin, il est possible de générer un nombre illimité d'exemples, ce qui permet de ne pas présenter plusieurs fois le même à un modèle lors de l'étape d'apprentissage.

L'utilisation d'images synthétiques peut également répondre à un autre besoin : celui d'analyser un réseau de neurones artificiel, afin de comprendre ses propriétés. En créant des données dont tous les paramètres sont connus, il est possible d'analyser le comportement du modèle et d'en tirer des enseignements pour l'entraîner de façon optimale sur des données classiques. C'est ce que font Peng *et al.* (2014) : en modifiant la position et l'orientation de modèles 3D rendus avec différentes textures et sur différents arrière-plans, ils ont identifié les éléments face auxquels le détecteur d'objet est invariant. Leurs conclusions indiquent que le réseau de neurones testé est invariant à la texture, la couleur et la position de l'objet. Ces éléments doivent donc être montrés au réseau de façon la plus exhaustive possible afin qu'il apprenne des caractéristiques vraiment discriminantes ; ce sont donc ces éléments qu'il est important de synthétiser.

Si des images réelles sont disponibles en petite quantité, elles peuvent être mises à profit de différentes façons. Tout d'abord, elles peuvent être simplement utilisées pour spécialiser le réseau après l'entraînement avec des images synthétiques, lorsqu'elles sont accompagnées d'annotations (Tremblay *et al.*, 2018). Elles peuvent également être utilisées avec une approche qui ne requiert pas d'annotations ; par exemple, l'adaptation de domaine consiste à utiliser une petite quantité d'images du domaine cible pour adapter un modèle préalablement entraîné sur le domaine source (Sun et Saenko, 2014; Long *et al.*, 2015; Ganin et Lempitsky, 2015; Nogues *et al.*, 2018). En manipulant les caractéristiques apprises, le modèle obtient ainsi une performance similaire sur le domaine cible (les images réelles) que sur le domaine source (les images synthétiques). Enfin, les méthodes génératives utilisent une petite quantité d'images réelles pour entraîner un réseau de neurones à générer une grande quantité d'images similaires, afin d'augmenter la quantité d'images d'entraînement disponible. L'un des réseaux de neurones les plus connus pour cela est StyleGAN (Karras *et al.*, 2019), ainsi que ses évolutions ultérieures StyleGAN2 (Karras *et al.*, 2020) et StyleGAN3 (Karras *et al.*, 2021). D'autres architectures basées sur StyleGAN ont également vu le jour, comme DatasetGAN (Zhang *et al.*, 2021b) qui permet de générer les cartes de segmentation associées aux images synthétiques, élargissant ainsi le champ d'application des images générées par des GANs au-delà de la simple classification. Un état de l'art récent et exhaustif des méthodes de génération d'images synthétiques peut

être trouvé dans le livre de Nikolenko (2021).

D'un point de vue pratique, les images synthétiques peuvent être générées à la volée durant l'entraînement, ce qui élimine le besoin de les stocker en mémoire et permet de créer de nouvelles images tout au long de l'entraînement, sans jamais les réutiliser. Cependant, cette génération « en ligne » des images d'entraînement empêche les expériences menées d'être reproduites, ce qui n'est pas une caractéristique désirée dans un contexte de recherche. Cela augmente aussi significativement la durée de l'entraînement. Nous avons donc travaillé avec des images générées une seule fois et stockées en mémoire pour les besoins de cette thèse, et proposé les deux approches à l'entreprise DEMS pour le déploiement en contexte industriel.

3.1.1 Images 2D augmentées

Parmi les différents types d'images synthétiques, on distingue les images générées à partir d'un environnement 3D des images dans lesquelles différents éléments sont assemblés, réels ou synthétiques. Ce sont ces dernières que l'on appelle des « images 2D augmentées », car elles sont créées par composition d'éléments synthétiques positionnés sur une image d'arrière-plan réelle. Lorsqu'une ou plusieurs photos des objets à identifier existent, une méthode directe consiste à découper cet objet, et à le coller sur différents arrière-plans (Dwibedi *et al.*, 2017; Georgakis *et al.*, 2017). Simple en apparence, cette approche demande toutefois de découper les objets d'intérêt de façon précise (ou bien de les acquérir sur un fond uni), éventuellement de définir des zones cohérentes pour coller l'objet dans la scène, et de tester différentes façons de fondre les bords de l'objet dans l'arrière-plan (Georgakis *et al.*, 2017). Si un modèle numérique de l'objet est disponible, tel qu'une visualisation 2D ou 3D générée par un logiciel, alors l'étape de découpe manuelle de l'objet peut être évitée. Cette approche est celle couramment utilisée lorsque des images réelles sont en nombre insuffisant pour apprendre correctement, ou lorsque l'on souhaite éliminer toute étape manuelle. En particulier lorsque la tâche à résoudre est celle de classification, il est possible de récupérer des modèles 3D correspondant à des classes spécifiques dans les bases de données publiques, puis de générer un rendu 2D de ces modèles 3D afin de les coller sur l'arrière-plan choisi (Sun et Saenko, 2014; Hinterstoisser *et al.*, 2018; Sampaio *et al.*, 2021). Plusieurs travaux étudient les caractéristiques de ces rendus 2D pour une reconnaissance optimale dans des images réelles, tels que la texture ou le niveau de réalisme, en quelle quantité ces images doivent être fournies, ou encore dans quel ordre elles doivent être générées (Peng *et al.*, 2015; Sarkar *et al.*, 2017; Hinterstoisser *et al.*, 2019; Hofer *et al.*, 2021). Alhaija *et al.* (2018) proposent une méthode efficace de génération d'objets synthétiques dans des images réelles en se basant sur les principes de la réalité augmentée, dans laquelle les objets virtuels se fondent dans le réel. Si cette approche crée des images hautement réalistes, elle nécessite toutefois qu'un utilisateur identifie des zones cohérentes dans les images réelles afin d'y placer les objets synthétiques.

Les images 2D augmentées trouvent des applications au-delà de la tâche de classification, grâce à la possibilité de produire les annotations de son choix : des boîtes englobantes pour la détection d'objets, des masques binaires pour la segmentation (Alhaija *et al.*, 2018), ou encore la position et l'orientation des objets en trois dimensions pour l'estimation de pose (Su *et al.*, 2015; Hofer *et al.*, 2021).

3.1.2 Scènes 3D

Afin d'obtenir des images synthétiques dans lesquelles les positions des objets sont cohérentes avec l'environnement, une direction de recherche a été de développer des environnements virtuels tels que ceux des jeux vidéo (Gaidon *et al.*, 2016; Song *et al.*, 2017; Jalal *et al.*, 2019; Leonardi *et al.*, 2022). Ils visent en particulier à combler l'une des lacunes des images 2D augmentées, à savoir la différence entre les objets d'intérêt et l'arrière-plan. Dans ces environnements, la plupart des paramètres sont ajustables, comme les objets visibles, le niveau de réalisme, la luminosité ou encore le point de vue, ce qui permet de faire varier l'apparence des images synthétiques de façon cohérente sur l'ensemble de l'image. Les annotations peuvent également être générées très facilement, toutes les informations liées aux objets, à la scène et à la caméra étant connues et contrôlables.

La génération de scènes entières présente un second avantage : la possibilité de simuler l'image de profondeur associée à une image RGB. En effet, avec une image 2D augmentée, il est difficile de simuler une image de profondeur qui soit cohérente entre l'objet synthétique ajouté, et l'arrière-plan réel. Bien que cela ait été proposé (Thalhammer *et al.*, 2019), une connaissance fine de la scène et des objets ajoutés est nécessaire, afin d'obtenir une image RGB-D cohérente pour un réseau de neurones. Au contraire avec une scène 3D, il est facile de mesurer la distance entre la caméra et chacun des objets, afin d'obtenir une image de profondeur parfaitement cohérente avec l'image RGB.

Cependant, la génération de scènes photoréalistes prend plus de temps que la génération de quelques objets seulement, ce qui a conduit les chercheurs à proposer des solutions alternatives. Deux points de vue s'affrontent donc : générer des vues 2D photoréalistes (Langlois *et al.*, 2018), ou utiliser la randomisation de domaine (Tobin *et al.*, 2017; Tremblay *et al.*, 2018). Alors que la génération d'images photoréalistes peut être coûteuse en temps de calcul, la randomisation de domaine mise sur la quantité plutôt que la qualité des vues générées. En effet, cette méthode consiste à générer des images extrêmement variées, notamment dans leur texture ou couleur, même si elles ne sont absolument pas réalistes. L'objectif est de faire apprendre au modèle suffisamment de variations pour que les images réelles apparaissent comme une variation supplémentaire, à laquelle le réseau sera robuste. Au-delà des objets d'intérêt présents dans chaque image, les éléments qui peuvent être randomisés sont par exemple : le nombre et le type d'objets distracteurs (objets que l'on ne veut pas identifier), la texture des objets, l'arrière-plan (avec différents environnements, une couleur unie ou un motif artificiel), les sources lumineuses, les positions des objets, etc. Plusieurs travaux comparent les deux approches, et proposent finalement de les combiner pour obtenir de meilleurs résultats. Nogues *et al.* (2018) comparent les résultats obtenus avec des images synthétiques simples, randomisées avec des couleurs, randomisées avec des textures, et transférées dans le domaine réel avec un réseau génératif (Figure 3.1). Tremblay *et al.* (2018) appliquent uniquement des textures non réalistes, ainsi que des éléments aléatoires dans l'environnement, contrairement à la pratique courante de générer des scènes urbaines réalistes (Figure 3.2). Dans les deux cas, les images synthétiques randomisées ont permis une amélioration des résultats pour un coût de préparation du jeu de données limité, et bien inférieur à l'acquisition et annotation d'images réelles, ou à la génération d'images extrêmement réalistes. Pour (Nogues *et al.*, 2018), c'est même la combinaison de différents types d'images synthétiques, présentant différents niveaux de réalisme, qui permet d'obtenir la performance la plus élevée sur les images réelles.

(a) Simple (b) Randomisée (c) Randomisée avec des textures (d) Photoréaliste avec un GAN

FIGURE 3.1 – Exemples d'images synthétiques issues des travaux de Nogues *et al.* (2018).

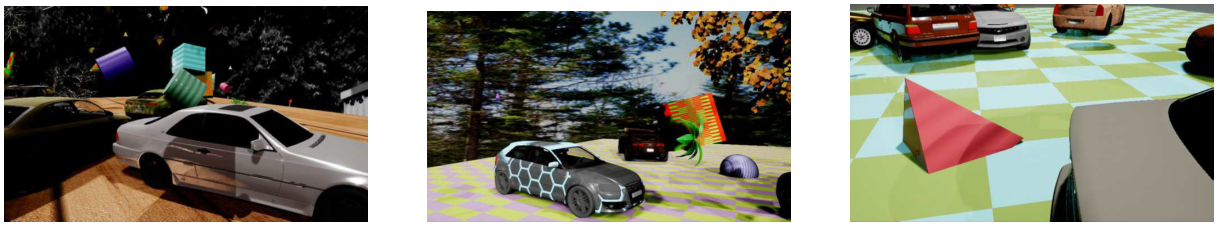


FIGURE 3.2 – Exemples d'images synthétiques randomisées issues des travaux de Tremblay *et al.* (2018).

3.1.3 Limitations des images synthétiques

L'utilisation d'images synthétiques présente beaucoup d'avantages et a permis l'amélioration de la performance dans de nombreuses applications. Cependant, elles incluent une problématique supplémentaire à prendre en compte : l'écart de domaine entre les images synthétiques et réelles, aussi appelé écart à la réalité (*reality gap* en anglais). En effet, lorsqu'un réseau de neurones artificiel est entraîné, il apprend une distribution des images vues pendant l'entraînement. Si les images de test sont très différentes, la distribution n'est plus la même, et ne correspond plus à ce que reconnaît le réseau. Ainsi, un réseau entraîné sur des images synthétiques obtient généralement une performance plus basse sur des images réelles. En fonction de l'application considérée, il est parfois nécessaire de réduire cet écart de domaine afin que le réseau de neurones entraîné sur des images synthétique soit suffisamment efficace sur des images réelles.

3.2 Images 2D augmentées égocentriques

Dans cette partie, nous présentons la méthode que nous avons proposée pour créer des images synthétiques pour la détection d'objets dans des images réelles égocentriques. Alors que la plupart des approches créent des images correspondant à une caméra fixe, notre contexte industriel impose que les images réelles soient acquises avec une caméra mobile, positionnée au niveau de la tête de l'utilisateur. Ces images induisent un certain nombre de difficultés, comme présenté dans le chapitre 1, que nous rappelons ici : les objets ne sont pas complètement visibles dans le champ de la caméra, les images présentent des occultations (entre les objets, ou occasionnés par les bras de l'utilisateur), il y a beaucoup de flou et de changements de luminosité dans les images. Nous nous sommes donc inspiré des méthodes de génération d'images 2D augmentées afin de proposer notre propre processus, qui prend en compte l'aspect égocentrique des images réelles sur lesquelles le réseau de neurones devra ensuite généraliser.

La génération d'images 2D augmentées égocentriques se déroule en trois étapes principales :

- Génération de vues 2D des objets à partir des modèles 3D
- Composition des vues 2D avec des arrière-plans réels
- Augmentation des images composées

Ces étapes sont représentées sur la Figure 3.3.

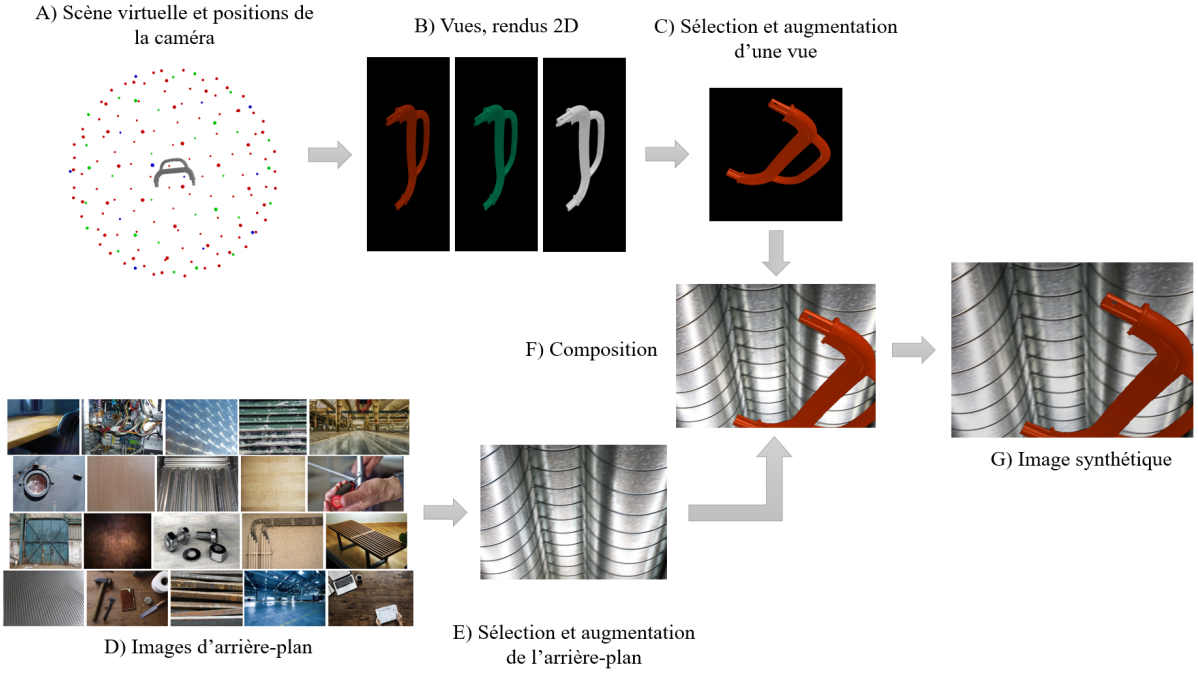


FIGURE 3.3 – Processus de génération des images 2D augmentées égocentriques.

3.2.1 Rendu des vues 2D

Pour chaque objet d'intérêt, nous disposons d'un modèle 3D sans texture, que nous visualisons à l'origine d'une scène 3D vide (Figure 3.3-A). Nous utilisons pour cela la librairie Python *trimesh* [36] qui se base sur la bibliothèque logicielle OpenGL et permet la manipulation de modèles de type *mesh*, c'est-à-dire dont l'architecture est définie par un maillage triangulaire. Une caméra virtuelle est placée à différentes positions dans cet espace 3D, avec son axe optique en direction de l'objet. Contrairement à d'autres travaux (Klein et Murray, 2009), nous ne faisons pas varier les paramètres de la caméra, et nous ne simulons pas non plus les caractéristiques physiques des caméras pouvant altérer les images.

Pour obtenir des positions de caméra distribuées régulièrement, nous appliquons la méthode proposée par Hinterstoisser *et al.* (2008) et représentée sur la Figure 3.4, dans laquelle les points de vue sont échantillonnés à partir d'un icosaèdre. Les 20 faces triangulaires de ce volume régulier à 12 sommets sont divisées de façon itérative en 4 nouveaux triangles, dans une étape d'affinement ; ces nouveaux sommets correspondent à un échantillonnage plus précis de l'espace 3D. La taille du modèle 3D n'est pas modifiée ici mais

36. https://trimsh.org/trimesh.htm

plus tard dans le processus. Alors que pour résoudre une tâche d'estimation de pose, des milliers de points de vue sont nécessaires (Wohlhart et Lepetit, 2015; Kehl *et al.*, 2017), ce n'est pas le cas pour la détection d'objets. Cette méthode d'échantillonnage permet d'obtenir 12, 42 ou 162 points dans l'espace avec respectivement 0, 1 ou 2 niveaux d'affinement (0 correspondant aux sommets initiaux de l'icosaèdre). Une vue 2D, ou rendu, correspond à l'image de cette scène depuis le point de vue de la caméra. Nous générons donc pour chaque objet des vues 2D en plaçant la caméra sur chacun des sommets obtenus après affinement. Les vues résultantes couvrent deux des trois degrés de liberté de l'objet, correspondant aux deux rotations hors plan (Figure 3.3-A). Pour obtenir des vues de l'objet couvrant les trois degrés de liberté, nous faisons tourner l'objet dans le plan dans une étape ultérieure.

Nous ajoutons des variations supplémentaires en changeant la couleur des modèles 3D : nous choisissons manuellement une liste de 12 couleurs, parmi lesquelles 5 sont choisies au hasard pour chaque position de caméra (Figure 3.3-B). L'illumination est gérée par une simple source lumineuse ponctuelle, émettant dans toutes les directions et créant des réflexions pour certaines couleurs et certains points de vue. Les objets sont rendus avec un fond noir permettant de générer très facilement un masque binaire pour différencier les pixels associés à l'objet des pixels du fond, et ainsi l'extraire automatiquement.

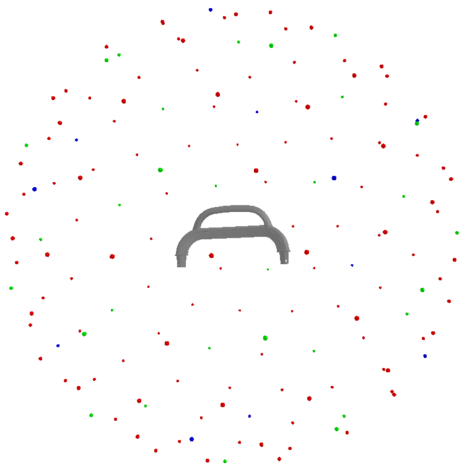


FIGURE 3.4 – Positions de la caméra virtuelle autour de l'objet. Les 12 points bleus correspondent au niveau d'affinement 0, les 30 points verts au niveau 1, et les 120 points rouges au niveau 2.

3.2.2 Composition des images

De précédents travaux ont étudié l'influence de l'image d'arrière-plan pour une tâche de classification (Peng *et al.*, 2015; Sarkar *et al.*, 2017). Ils ont montré que de meilleurs résultats sont obtenus lorsque l'arrière-plan est une image réelle, ce qui nous a encouragé à adopter cette approche. Nous définissons donc l'étape de composition des images comme l'assemblage d'une vue 2D synthétique d'un objet d'intérêt avec une image réelle. C'est lors de cette étape que l'annotation de l'image est automatiquement créée. Les travaux précédents utilisaient comme arrière-plans des images de bases de données publiques, en sélectionnant celles correspondant à des mots-clés tels que « scène d'intérieur », et en éliminant manuellement les images montrant des objets similaires aux objets d'intérêt. Cependant, les bases de données usuelles ne présentent pas de scènes correspondant à

notre application, c'est-à-dire avec des éléments faisant référence à une usine ou un site industriel, c'est pourquoi nous avons sélectionné 50 images manuellement à partir de banques d'images libres de droit, correspondant à des mots-clés (en anglais) tels que « industrial », « manufacturing », ou encore « workplace » (Figure 3.3-D).

Les images synthétiques sont assemblées en suivant la procédure de la Figure 3.3. En premier lieu, une image d'arrière-plan est sélectionnée au hasard et augmentée avec des opérations classiques : rognage des bords de l'image et redimensionnement, modification des valeurs des pixels, normalisation de contraste aléatoire, symétries horizontale et verticale, et enfin flou de mouvement (Figure 3.3-E). Ensuite, une vue est sélectionnée, redimensionnée au hasard, et sa position sur l'arrière-plan est définie aléatoirement également (Figure 3.3-C). Cette vue peut être positionnée jusqu'à un tiers en dehors de l'image, afin de simuler les objets partiellement visibles lorsqu'ils sont manipulés et vus depuis une caméra égocentrique. La vue est également augmentée avec une rotation aléatoire dans le plan et une multiplication de la valeur des pixels, avant d'être projetée sur l'arrière-plan augmenté (Figure 3.3-F). Du bruit Gaussien est ajouté à toute l'image afin de lisser les contours de l'objet, et rendre plus harmonieuse l'image obtenue (Figure 3.3-G).

Pour terminer de réduire l'écart de domaine synthétique-réel, nous simulons à nouveau le mouvement avec soit du flou Gaussien, soit du flou de mouvement (Figure 3.5-3.5b)). Enfin, nous ajoutons des ombres en créant des formes noires semi-transparentes, que nous superposons sur les images de façon aléatoire (Figure 3.5). Bien que ces ombres ne soient pas réalistes, elles ajoutent des variations conséquentes à l'aspect des objets, notamment en évitant qu'un objet ait exactement la même couleur sur toute sa surface.

Le flou ajouté sur l'image finale ainsi que l'application des ombres ont ensuite fait l'objet d'une étude, afin d'identifier dans quelle mesure ces éléments sont utiles pour la détection d'objets dans des images réelles égocentriques (Chapitre 4).

(a) Images synthétiques

(b) Images avec du flou

(c) Images avec des ombres

FIGURE 3.5 – Exemples d'images synthétiques 2D augmentées RUSPA, avec du flou et des ombres.

3.2.3 RUSPA RGB avec la méthode proposée

Avec la méthode proposée dans la partie précédente, nous avons généré des images d'entraînement synthétiques pour le jeu de données RUSPA de l'entreprise DEMS, afin d'entraîner un détecteur d'objets. Pour cela, nous disposions initialement des 5 modèles 3D issus des données de conception de ce siège de bus, correspondant aux parties suivantes : la poignée, le dossier, l'assise, le profil droit et le profil gauche. La première étape a été le

nettoyage des modèles 3D, dont les surfaces n'étaient pas normalisées. En effet, certains triangles qui composent le maillage des modèles étaient dessinés « à l'envers » (le vecteur normal pointe du mauvais côté), ce qui appliquait la couleur à l'intérieur de l'objet et rendait certaines zones transparentes lorsque les objets étaient rendus (Figure 3.6). Après ce nettoyage manuel avec le logiciel Blender, nous avons appliqué les étapes de la méthode décrite précédemment. Dans un premier temps, nous avons utilisé 42 points de vue de caméra, mais les résultats n'étaient pas satisfaisant : le réseau entraîné reconnaissait bien les objets dans certaines positions, mais ne les reconnaissait plus lorsque ceux-ci bougeaient légèrement. Nous en avons déduit qu'un nombre insuffisant de positions composait les données d'entraînement, et nous avons choisi d'utiliser 162 positions différentes pour la caméra virtuelle. Chaque objet se voit appliquer 5 couleurs différentes par point de vue, ce qui donne un total de 810 images par classe. Nous avons ensuite appliqué l'étape de composition des images en définissant 15 rotations dans le plan, différentes pour chaque objet. Au total, chaque objet apparaît dans 2430 images avec un point de vue, une couleur et une orientation différents à chaque fois. Nous avons sélectionné 2000 vues par classe, de façon aléatoire. Toutes les images synthétiques créées ici ne comprennent qu'un objet par image, laissant la simulation des occultations pour une analyse ultérieure.

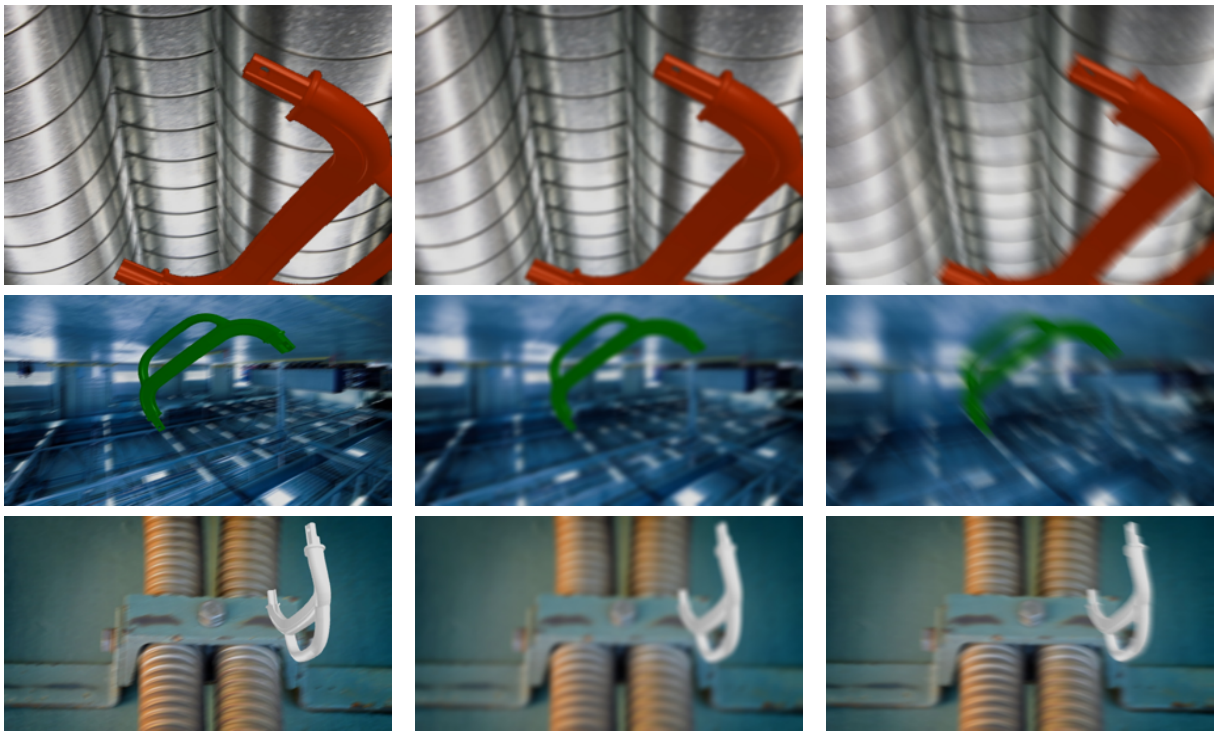
FIGURE 3.6 – Exemple de modèle 3D dont les faces sont à l'envers, ce qui empêche la couleur de s'appliquer correctement et crée un « trou » dans l'objet.

En ce qui concerne les paramètres de l'augmentation d'image appliquée, le flou Gaussien a une valeur moyenne de 0 et un écart-type variant entre 5 et 12. Le flou de mouvement est obtenu avec un noyau de taille variant entre 10 et 80 pixels, et un angle et une direction de mouvement aléatoires. Ces paramètres sont choisis au hasard pour chaque image. La Figure 3.7 montre des exemples d'images obtenues lorsque différents types de flou sont appliqués.

3.3 Scènes 3D pour des images synthétiques RGB-D

3.3.1 BlenderProc

Parmi les générateurs de jeux de données existants dans la littérature, nous avons choisi d'utiliser BlenderProc, que nous présentons ici. Proposé par Denninger *et al.* (2019), BlenderProc est un projet logiciel basé sur le moteur de rendu physique réaliste Blender, un



(a) Pas de flou (b) Flou Gaussien (c) Flou de mouvement

FIGURE 3.7 – Exemple d'images 2D augmentées du jeu de données RUSPA, avec plusieurs types de flou.

outil gratuit et très prisé des designers, artistes et animateurs 3D. Blender possédant une interface en langage Python, il est possible de créer une scène, d'y créer ou importer des modèles 3D ainsi qu'une caméra virtuelle, et de générer des images correspondant à diverses modalités, le tout depuis un script Python. La bibliothèque logicielle BlenderProc propose une série d'outils pour aider les chercheurs à générer leurs images synthétiques, à partir d'un fichier de configuration dans lequel sont indiqués les objets, les sources lumineuses ou encore les paramètres de la caméra, ainsi que toutes leurs positions et le nombre d'images à générer. Un avantage de ce fichier de configuration est qu'il permet d'appliquer des valeurs de façon aléatoire à un grand nombre de propriétés, en ne spécifiant que les limites d'un intervalle par exemple. Cela permet d'appliquer le principe de la randomisation de domaine, en créant des images variées dont les paramètres sont choisis au hasard lors de la génération des images.

Alors que de nombreux exemples existent déjà pour différentes bases de données publiques et tâches à résoudre (segmentation, estimation de pose ou de profondeur, etc.), BlenderProc est un logiciel *open source* permettant à quiconque de l'étendre en ajoutant des modules logiciels supplémentaires. Dans la suite de cette partie, nous décrivons notre méthode pour générer une scène avec BlenderProc, puis l'application à RUSPA. Nous présentons en détails les modules principaux de BlenderProc dans l'annexe A.

3.3.2 Génération d'une scène 3D

Un rendu 2D est obtenu en plaçant une caméra virtuelle et une source lumineuse dans une scène 3D. La simulation des rayons lumineux permet d'obtenir l'image de la scène du point de vue de la caméra. Ainsi, la création d'une scène 3D demande plusieurs étapes :

- la construction de la scène
- la définition des sources lumineuses
- la définition de la caméra

La génération d'un jeu de données annotées avec BlenderProc comprend quelques étapes supplémentaires :

- la variation des paramètres pour obtenir des images différentes
- la définition du type d'annotations désiré

Avec BlenderProc, toutes ces étapes peuvent être réalisées dans le fichier de configuration. Dans un premier temps, la scène est construite par l'insertion d'objets, soit par génération de primitives, c'est-à-dire des formes géométriques simples telles que des sphères ou des cubes, soit par importation de modèles 3D pré-existants. Ces objets peuvent être les objets que l'on souhaite détecter, d'autres objets servant à construire un environnement (comme des tables et chaises pour créer un intérieur), ou bien des objets distracteurs. Ils sont identifiés par des attributs créés par l'utilisateur, qui pourront ensuite servir à les sélectionner ou non au moment d'appliquer certains paramètres. Afin que ces objets ne flottent pas dans la scène, il est possible d'ajouter des surfaces pour faire office de murs et de sols, et de simuler la gravité et les collisions pour qu'ils se placent de façon réaliste. En plus d'ajouter de la texture à l'arrière plan, les surfaces permettent de limiter la taille de la scène pour générer une carte de profondeur plus réaliste. A cette scène sont ajoutées une ou plusieurs sources lumineuses, dont le type, la couleur, l'intensité et la position sont au choix de l'utilisateur. Enfin, une caméra est ajoutée à la scène. Comme pour la plupart des logiciels de rendu, Blender permet d'ajuster un très grand nombre de paramètres de la caméra comme sa définition ou sa distance focale. Il est ainsi possible de modéliser précisément une caméra réelle connue, afin que les images générées soient les plus proches possibles des images réelles capturées par cette caméra. Une fois les éléments assemblés dans la scène, les valeurs initiales de leurs paramètres sont définies. Il peut s'agir des couleurs et textures, comme des tailles et positions des objets.

Pour une scène donnée, BlenderProc permet de générer plusieurs rendus en spécifiant plusieurs positions pour la caméra. La scène reste cependant identique sur ces différents rendus. Afin de la faire varier, il est nécessaire de définir des intervalles de valeurs pour les différents paramètres et de les choisir au hasard lors de la construction de la scène. Ainsi, chaque fois que le fichier de configuration est parcouru, une valeur différente est choisie. Il ne reste plus qu'à relancer plusieurs fois le script pour générer des images correspondant à des scènes différentes, et depuis différents points de vue.

BlenderProc propose différents types d'annotations : les masques de segmentation des objets, associés à des étiquettes de classe et d'instance ; les boîtes englobantes correspondant aux objets entiers ou seulement à leur partie visible dans l'image ; les positions des objets dans la scène 3D par rapport à la caméra... Le format dans lequel enregistrer les annotations et les images est également laissé au choix de l'utilisateur. Une fois la scène construite et les paramètres fixés, BlenderProc procède au rendu depuis chaque point de vue spécifié, en faisant varier les paramètres nécessaires, et enregistre les images et leurs annotations. Des paramètres associés à la méthode de rendu peuvent être ajustés afin de réduire le temps nécessaire à la génération des images, ce qui réduit également la qualité des images obtenues.

3.3.3 RUSPA RGB-D avec BlenderProc

Après avoir pris en main les exemples proposés, nous avons créé notre fichier de configuration afin de générer des images synthétiques RGB-D pour le jeu de données RUSPA. Comme conseillé dans la documentation, nous avons créé un fichier de configuration qui définit un petit nombre de positions de caméras, et nous relançons le script principal un grand nombre de fois jusqu'à obtenir le nombre d'images souhaité. Dans chaque image, les objets peuvent être cachés aléatoirement ; par conséquent, tous les objets n'apparaissent pas dans toutes les images. Le résultat est un ensemble de 50 000 images synthétiques RGB-D montrant les différentes objets à détecter dans des positions et orientations différentes, avec des occultations, reflets et ombres, ainsi qu'avec des couleurs et textures variées.

Dans chaque scène, les 5 objets de la base RUSPA sont chargés, et deux objets additionnels sont chargés comme distracteurs, tous avec des modules *ObjectLoader*. Ces deux objets distracteurs sont pris aléatoirement, l'un parmi les 30 modèles 3D du jeu de données T-LESS[37], et l'autre parmi les 15 modèles 3D du jeu LINEMOD[37]. Ensuite, deux modules *MaterialManipulator* choisissent une texture pour chacun des objets (RUSPA et distracteurs), en tirant aléatoirement une image dans un dossier. Ce dossier contient des images au format jpeg de 32 couleurs et de 43 textures de matériaux (bois, métal, pierre, textile, etc.) que nous avons téléchargées à partir d'une banque d'images publique[38] (Figure 3.8). Les objets RUSPA sont ensuite mis à l'échelle pour que leur unité corresponde à celle de la scène Blender (des millimètres), la méthode d'ajustement de la texture au maillage est sélectionnée, puis chaque objet est rendu invisibles ou non, indépendamment des autres. Ainsi, tous les objets n'apparaissent pas dans toutes les images. Cette propriété est définie avec un *sampler* de type *Value* qui fournit un booléen pour chaque objet. Les objets distracteurs sont également mis à l'échelle, et la même méthode d'ajustement de texture est choisie.

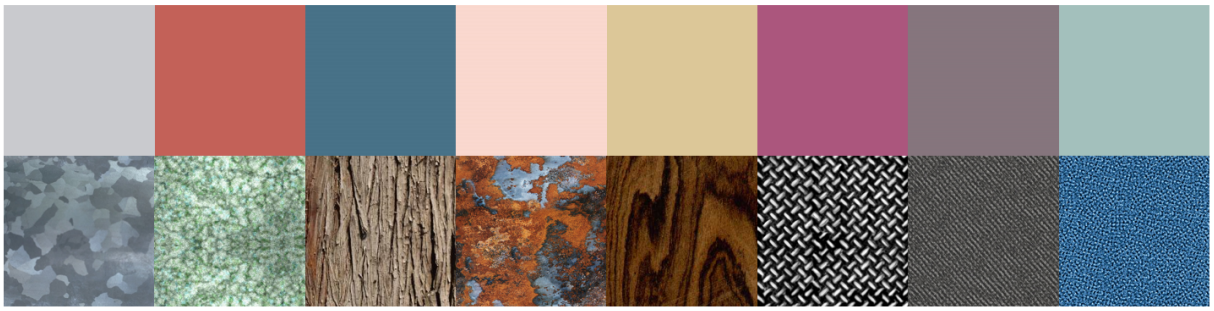


FIGURE 3.8 – Exemples de couleurs et textures appliquées sur les objets et surfaces de la scène dans BlenderProc.

Ensuite, la scène autour des objets est définie. Afin de ne pas risquer d'avoir des valeurs de profondeur « à l'infini », nous plaçons une surface plane horizontale entourée de 4 plans verticaux, qui forment les murs de la scène. Un dernier plan horizontal est placé au-dessus, pour servir de source lumineuse. Ces plans sont associés à la classe « arrière-plan », afin de ne pas être confondus avec des objets d'intérêt dans les annotations. Le matériau du plan supérieur est défini comme un émetteur de lumière, de couleur et intensité variables.

37. https://bop.felk.cvut.cz/datasets/
38. https://ambientcg.com/

Les autres plans sont associés à des textures, à nouveau choisies aléatoirement parmi les 75 couleurs et textures fournies. Les positions des objets sont modifiées avec un module *ObjectPoseSampler*, puis le module *PhysicsPositioning* est appelé pour définir les paramètres de la simulation.

Une source lumineuse ponctuelle est également ajoutée avec un module *LightSampler*, dont les paramètres de position et couleur sont choisis à nouveau aléatoirement dans un intervalle donné. L'intensité de cette source est fixée à 100, c'est principalement elle qui rend la scène visible. Le module *CameraSampler* définit les positions possibles pour la caméra, et contraint l'orientation afin d'orienter la caméra vers les objets d'intérêt. La définition est de 640 par 480 pixels, pour être en adéquation avec les images réelles RGB-D à disposition (voir section 5.3.2.1).

Enfin, les images sont générées par un *RgbRenderer* qui donne également la carte de profondeur associée, avec un intervalle de distances de 10cm à 10m (à nouveau, pour être cohérent avec les paramètres de la caméra de profondeur réelle utilisée, voir paragraphe 5.3.2.1). Les annotations sont enregistrées au format HDF5 avec un module *Hdf5Writer*.

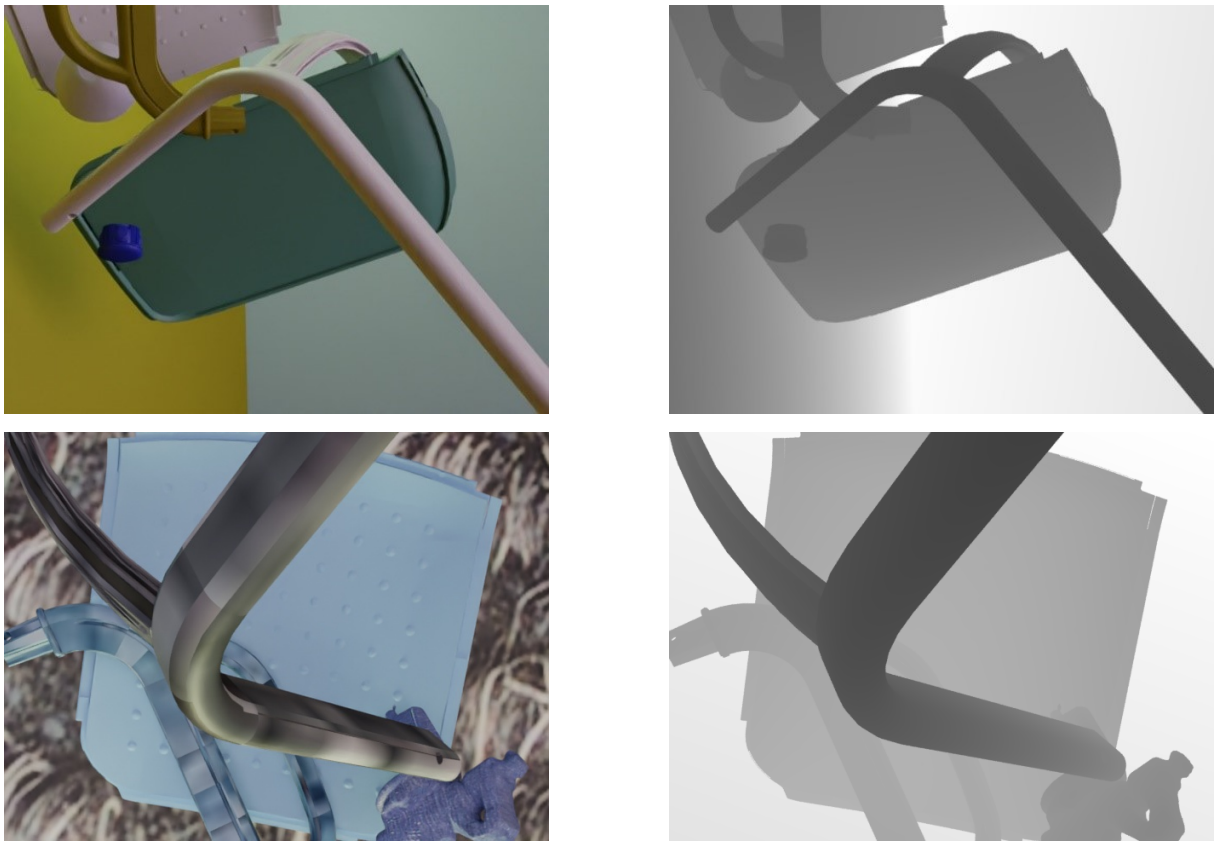


FIGURE 3.9 – Exemples d'images synthétiques RGB-D générées avec BlenderProc pour le jeu de données RUSPA.

Après avoir généré un peu plus de 50 000 images, nous avons écarté celles dans lesquelles les objets étaient trop peu visibles et identifiables, comme celles trop proches de la caméra mais présentant seulement un petit bout de l'objet dans le champ de l'image. Nous en avons gardé exactement 50 000 afin de suivre l'exemple des bases de données publiques T-LESS et LINEMOD, que nous avons utilisé par la suite. Des exemples de ces images sont visibles dans l'Annexe B.

3.4 Conclusion

De nombreuses méthodes existent afin de générer des images synthétiques pour l'apprentissage profond. En fonction des informations disponibles (modèles 2D ou 3D, textures) et du contexte, il est possible d'assembler simplement des images en faisant varier un grand nombre de paramètres. Pour créer des images RGB d'apparence égocentrique, nous avons choisi l'approche simple des images 2D augmentées, qui assemble un modèle 3D à n'importe quelle image d'arrière-plan réelle. Nous avons proposé d'ajouter une quantité importante de flou, ainsi que des ombres artificielles, afin de réduire l'écart à la réalité avec des images égocentriques. Ce premier jeu de données nous permet par la suite d'évaluer la capacité d'un réseau de neurones à identifier des objets dans des images réelles égocentriques, en étant entraîné uniquement avec des images synthétiques. Ces travaux ont été publiés lors de la conférence VISAPP en 2020 (Cohen *et al.*, 2020) et sont détaillés dans le prochain chapitre, avec notamment une étude ablative de certains éléments synthétiques. Pour aller plus loin, il est possible de compléter cette base en la complexifiant, par exemple avec plusieurs objets par image afin de simuler les occultations qui peuvent survenir lorsque les objets sont manipulés.

Ensuite, nous avons adopté une approche par génération de scène 3D afin de créer des images RGB-D synthétiques. Pour cela, nous avons étudié et adopté l'outil BlenderProc, qui permet de générer des images et leurs annotations à partir de modèles 3D et d'un fichier de configuration. Dans ces images, plusieurs objets d'intérêt ainsi que des objets distracteurs sont visibles, souvent partiellement hors de l'image comme dans les images égocentriques réelles. Les ombres artificielles ne sont pas ajoutées ici, car les différents objets de la scène 3D créent déjà des ombres réalistes sur les objets et des variations de teinte pour une même couleur ou texture. Les différents flous et bruits pour rendre les images plus réalistes seront ajoutées comme augmentation au moment de l'entraînement du réseau de neurones. Les images RGB de ces paires RGB-D sont utilisées dans le prochain chapitre pour évaluer une seconde architecture de réseau de neurones, toujours pour la détection d'objets. Cette architecture est étendue dans le Chapitre 5 aux images de profondeur.

Bien que plusieurs métriques aient été définies dans la littérature pour évaluer la qualité des images générées artificiellement (Salimans *et al.*, 2016; Heusel *et al.*, 2017), elles s'appuient sur la capacité d'un réseau pré-entraîné avec ImageNet à identifier les objets visibles dans les images, ou sur des caractéristiques extraites d'un tel réseau. Nos images ne montrant pas des objets du quotidien tels que ceux présents dans les images du jeu de données ImageNet, ces métriques ne sont pas adaptées. De plus, elles ne sont définies que pour les images RGB, et non pour les cartes de profondeur. Ainsi, nous évaluerons la qualité de nos images synthétiques par rapport à la performance obtenue par nos réseaux de neurones. Une image synthétique peut ne pas sembler réaliste pour l'œil humain, mais peut comporter les éléments nécessaires à un réseau de neurones pour généraliser à des images réelles, ce qui est notre objectif principal.

Chapitre 4

Détection d'instances d'objets dans des images couleur

Ce chapitre présente les expériences que nous avons menées pour détecter des objets industriels dans des images RGB, à partir des modèles 3D de ces objets. Nous appliquons d'abord le réseau YOLOv3 à la détection d'objets dans des images réelles égocentriques, entraîné avec une combinaison d'images réelles et synthétiques. Nous explorons différentes stratégies et hyperparamètres influant sur la performance d'un réseau de neurones, tels que le pré-entraînement et le *fine-tuning*, et nous étudions l'apport potentiel d'une petite quantité d'images réelles pendant l'entraînement. Nous analysons également les résultats obtenus en faisant varier l'apparence des images synthétiques utilisées pendant l'entraînement, en particulier l'ajout d'ombres et de flou. Dans un deuxième temps, nous décrivons les expériences menées avec un détecteur différent, le réseau Single-Shot Detector (SSD). Nous entraînons ce réseau avec des images entièrement synthétiques issues de scènes 3D, et sans aucune image réelle. Nous comparons la performance de plusieurs extracteurs de caractéristiques de la famille des MobileNets, et nous proposons une stratégie d'augmentation de données spécifique aux images synthétiques afin de réduire l'écart à la réalité et faciliter la généralisation du réseau de neurones aux images réelles.

Sommaire

4.1 Introduction

Dans notre contexte industriel, nous nous intéressons aux architectures d'apprentissage profond *single stage*, qui permettent de détecter des objets dans un temps très réduit et qui nécessitent peu de ressources de calcul et de mémoire. Alors que les détecteurs d'objets basés sur les réseaux de neurones sont aujourd'hui utilisés dans des applications pour le grand public, ils montrent cependant plusieurs limitations relatives à notre application. La première limitation est leur temps d'inférence : en étant trop lents, ces réseaux ne peuvent pas être appliqués à la prédiction d'objets dans un flux vidéo en temps réel dans un appareil mobile. La solution couramment adoptée est d'envoyer les images dans le *cloud* afin qu'une machine plus puissante effectue les prédictions, qui sont ensuite renvoyées sur l'appareil mobile, ce qui ajoute un délai dû à la transmission des données (Farasin *et al.*, 2020). Lorsque le réseau est exécuté en local, c'est-à-dire sur l'appareil, il ne peut en général pas traiter un flux d'images en temps réel car ses capacités de calcul sont limitées. De plus, la consommation énergétique est généralement très importante, ce qui empêche l'appareil de fonctionner suffisamment longtemps pour être utilisé dans un contexte de production lorsqu'il repose sur une batterie portable.

Ensuite, les détecteurs d'objets sont très efficaces pour des catégories d'objets pour lesquelles une très grande quantité d'images est disponible. Grâce aux jeux de données massifs et à l'explosion du nombre d'images disponibles sur Internet, les réseaux de neurones disposent d'une quantité suffisante de données pour apprendre à reconnaître les objets du quotidien : les personnes, les animaux, le mobilier d'intérieur comme les canapés et les chaises, ou encore les véhicules. En revanche, leur performance est limitée lorsqu'il s'agit d'objets spécifiques comme des produits industriels, pour lesquels peu d'images sont disponibles. On parle alors de détection d'instances : une catégorie correspond à un objet unique, et non à un groupe d'objets. L'entraînement demande de créer des données à partir d'un seul exemple, comme un modèle 3D de l'objet à reconnaître, ou de quelques images seulement.

Ces limitations nous concernent directement et ont particulièrement orienté les choix de notre approche. En effet, nous voulons détecter des objets très spécifiques pour lesquels peu d'images réelles peuvent être acquises, voire aucune, et l'inférence doit s'exécuter en temps réel, sur une plateforme embarquée sans connexion internet. Au début de cette thèse, deux détecteurs d'objets ont été identifiés comme les plus performants pour la détection d'objets en temps réel : YOLOv3 et SSD. Nous avons d'abord étudié YOLOv3, avant de poursuivre avec SSD pour sa flexibilité et la facilité à modifier son architecture.

4.2 Détection avec YOLOv3 et des images 2D augmentées

Dans un premier temps, nous avons étudié le modèle YOLOv3 entraîné sur le jeu de données RUSPA, dont les images synthétiques sont des images 2D augmentées. Au travers de plusieurs expériences, nous avons analysé la capacité du réseau à apprendre à reconnaître des objets d'intérêt à partir d'une petite quantité d'images réelles, à partir d'images synthétiques, ou bien avec une combinaison des deux. Les paragraphes qui suivent décrivent l'architecture et les différentes méthodes d'entraînement, le jeu de données utilisé, ainsi que les résultats obtenus.

4.2.1 Rappel de l'architecture de YOLOv3

Nous avons choisi comme détecteur d'objets le réseau YOLOv3, présenté dans la section 2.3.2 et représenté sur la Figure 2.24. Le modèle est composé d'un extracteur de caractéristiques spécifique aux réseaux de la famille YOLO, nommé Darknet-53 en raison de ses 53 couches de convolution. A partir de cet extracteur de caractéristiques, une première branche de détection produit des prédictions avec une définition assez réduite. Une deuxième branche combine les cartes caractéristiques de l'extracteur ainsi que de la première branche de détection, pour donner des prédictions à une échelle plus précise. Enfin, la troisième branche s'appuie à nouveau sur l'extracteur de caractéristiques et la deuxième branche, afin d'obtenir une troisième série de prédictions. Alors que l'extracteur de caractéristiques fournit des informations sémantiques mais peu de définition spatiale, ces trois niveaux de détection permettent d'améliorer la précision spatiale sans perdre d'information sémantique. Nous nous référons à ces branches comme « le détecteur » ou « tête de détection », par opposition à l'extracteur de caractéristiques.

4.2.2 Stratégies d'entraînement de YOLOv3

Suivant la pratique la plus courante dans la littérature, nous utilisons YOLOv3 pré-entraîné sur le jeu de données ImageNet, ce qui nous laisse le choix de plusieurs stratégies d'entraînement. En effet, comme la plupart des détecteurs d'objets, YOLOv3 se compose d'un extracteur de caractéristiques et d'un détecteur d'objets, qui peuvent être optimisés différemment. Il est donc possible d'entraîner le réseau de bout-en-bout avec nos données, ou bien de n'entraîner que la tête du réseau en figeant les poids de l'extracteur de caractéristiques.

Les recommandations de Chu *et al.* (2016) à ce sujet sont d'utiliser un maximum de couches pré-entraînées, puis de réaliser le *fine-tuning* du réseau en entier avec les images du jeu de données d'intérêt. Cependant, ces travaux ne se sont intéressés qu'à des images réelles, et ne considèrent pas le problème de l'écart de domaine entre les images synthétiques et réelles. Dans ce cas précis, Hinterstoisser *et al.* (2018) ont montré de meilleurs résultats en figeant les poids pré-entraînés sur ImageNet, et en n'optimisant que les poids spécifiques à la tâche d'intérêt avec les images synthétiques. En revanche, Tremblay *et al.* (2018) ont obtenu les résultats inverses, avec une bien meilleure performance lorsque les poids ne sont pas figés mais entraînés de bout-en-bout avec des images synthétiques. Parmi leurs résultats, ils montrent également qu'avec un réseau pré-entraîné sur ImageNet, la performance ne s'améliore plus en augmentant le nombre d'images synthétiques au-delà de 10 000. Face à ces conclusions divergentes, nous avons choisi de comparer les différentes approches avec notre réseau et jeu de données.

Le Tableau 4.1 introduit les notations que nous avons choisi pour identifier les différents réseaux, en fonction de leur pré-entraînement et entraînement. Lorsque le réseau est entraîné de bout-en-bout, il est noté « Net » ; lorsque seule la tête de détection est entraînée, le réseau est appelé « Det ». Le préfixe « Real » est accolé lorsque ce réseau est entraîné avec des images réelles, et « Synth » pour des images synthétiques. « Synth » est remplacé par « Blur » si les images synthétiques sont floutées, « Shadows » si des ombres sont ajoutées, et « BS » si elles contiennent du flou et des ombres (ces images sont présentées dans la section 3.2.2, et illustrées sur la Figure 3.5). « +FT » est ajouté à la fin lorsque le réseau est entraîné avec des images synthétiques, puis avec des images réelles comme *fine-tuning* du détecteur. Enfin, nous avons également analysé la performance en mélangeant des images réelles aux images synthétiques pendant l'entraînement ; le réseau

entraîné de cette façon est noté avec le préfixe « SynthReal ».

TABLEAU 4.1 – Notation et caractéristiques des entraînements du réseau YOLOv3. « Net » correspond à un entraînement de bout en bout, « Det » à l'entraînement du détecteur uniquement, *Fine-tuning* à l'entraînement du détecteur avec des images réelles après un premier entraînement avec des images synthétiques.

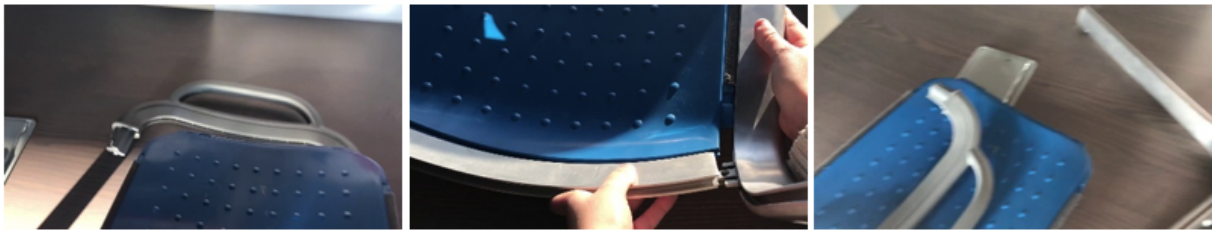
Nom	**Type d'images**	**Flou, ombres ?**	***Fine-tuning***
Net_Real	Réelles	✗	✗
Det_Real	Réelles	✗	✗
Net_Synth	Synthétiques	✗	✗
Det_Synth	Synthétiques	✗	✗
Det_Shadows	Synthétiques	Ombres	✗
Det_Blur	Synthétiques	Flou	✗
Det_BS	Synthétiques	Ombres et flou	✗
Net_Synth+FT	Synthétiques	✗	✓
Det_Synth+FT	Synthétiques	✗	✓
Det_Blur+FT	Synthétiques	Flou	✓
Det_SynthReal	Synthétiques et Réelles	✗	✗

4.2.3 Le jeu de données RUSPA

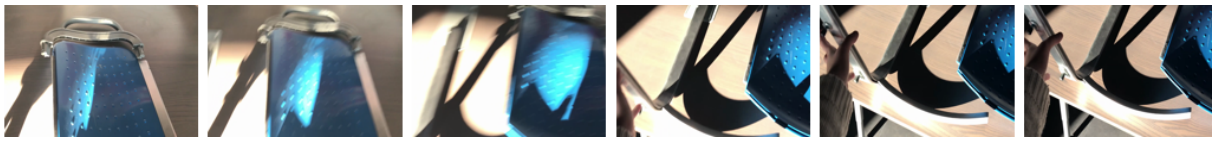
Disposant des modèles 3D et des pièces réelles correspondant aux cinq parties du siège de bus RUSPA (notre cas d'usage industriel), nous avons créé un jeu de données synthétique pour l'entraînement et un jeu de données réel afin d'évaluer le réseau de neurones. Nous avons généré les images synthétiques en suivant notre méthode de création d'images 2D augmentées présentée dans la section 3.2. Suivant les recommandations associées au réseau YOLOv3, nous avons créé 2000 images par classe, soit 10 000 images d'entraînement, ne comprenant ni le flou ni les ombres de la dernière étape du processus. Ensuite, pour la moitié de ces images, nous avons créé une version avec du flou (soit du flou Gaussien, soit du flou de mouvement), et une version avec des ombres synthétiques, dans le but d'étudier l'influence de ces deux éléments sur la performance du détecteur d'objets.

Afin d'évaluer la capacité de généralisation aux images réelles du réseau, nous avons acquis et annoté manuellement des images des objets d'intérêt (Figure 4.1). Nous avons rassemblé un total de 392 images avec trois smartphones différents. Ces images comprennent des photos, des images extraites de vidéos courtes filmées à la main de façon classique, ou des images extraites de vidéos égocentriques. Dans les deux premiers cas, les images montrent les objets centrés et posés sur une table, entrant complètement dans le champ de la caméra. Dans les vidéos, les mains de l'opérateur sont libres afin de manipuler l'objet, et l'on peut voir l'assemblage des pièces du siège de bus. Nous avons sélectionné des images espacées dans la vidéo, afin de montrer les objets sous différentes conditions de luminosité, de flou, et avec différentes orientations. Il est à noter cependant que ces images ne présentent pas les objets de façon exhaustive, et que les arrière-plans sont assez similaires sur toutes les images. Dans une même vidéo, par exemple, les images successives sont parfois presque identiques (Figure 4.1c) ; on y observe bien les difficultés liées aux images égocentriques et mentionnées précédemment, telles que les variations de luminosité, le flou, et les objets partiellement visibles.

(a) Photos acquises avec un smartphones.



(b) Images issues de vidéos égocentriques.



(c) Images successives d'une vidéo égocentrique.

FIGURE 4.1 – Exemples d'images réelles acquises pour constituer le jeu de données RUSPA. Plus d'images dans l'Annexe B.

Le Tableau 4.2 indique le nombre d'images dédiées à l'entraînement, à la validation et au test. Dans les images réelles, toutes les classes sont représentées dans les mêmes proportions (environ 20 %), excepté le dossier (environ 30 %) et le siège (environ 10 %).

TABLEAU 4.2 – Répartition des images du jeu de données RUSPA pour l'entraînement, la validation et le test.

	Entraînement	**Validation**	**Test**
Images synthétiques RUSPA RGB	6650	1675	1675
Images réelles	131	125	136

4.2.4 Expériences et résultats

Nous présentons ici les détails d'implémentation de nos expériences, ainsi que les résultats obtenus. Ces expériences visent à évaluer si des images synthétiques peuvent remplacer des images réelles égocentriques dans le cas des objets industriels, et dans quelles conditions. Nous évaluons également la qualité des images d'entraînement synthétiques égocentriques générées avec la méthode que nous avons proposée, via la capacité du réseau YOLOv3 à généraliser à des images réelles.

4.2.4.1 Procédure d'évaluation

Nous analysons la performance du réseau YOLOv3 avec deux métriques, la précision moyenne (mAP) et l'intersection sur l'union (IoU), décrites dans la section 2.2.3. Ici, la

mAP est calculée avec la méthode des 11 points de la courbe précision-rappel, et une détection est considérée positive pour l'IoU si elle se superpose de 50% ou plus à un objet (IoU@50).

Les images synthétiques sont réparties en trois sous-ensembles, dont deux sont utilisés pour l'entraînement, et le troisième est à nouveau divisé en deux pour la validation et le test. Nous appliquons un schéma d'entraînement par validation croisée avec trois sous-ensembles (*3-fold cross-validation*), afin que chaque sous-ensemble soit utilisé à la fois pour entraîner et pour tester le modèle. Les résultats décrits ci-dessous correspondent à la moyenne et à l'écart-type des trois performances obtenues. Pour les images réelles, la même approche est adoptée : nous répartissons les images dans trois sous-ensembles, mais cette fois en regroupant les images issues d'une même vidéo. En effet, des images provenant d'une même vidéo présentent des conditions d'illumination, d'arrière-plan, de point de vue, mais également de disposition des objets très similaires. Une bonne performance du réseau dans ce cas pourrait indiquer qu'il a appris par cœur la disposition de la scène pendant l'entraînement, sans être capable de généraliser correctement à d'autres images. Après avoir réparti les images issues de vidéos dans trois sous-ensembles, nous ajoutons les images restantes de façon à équilibrer le nombre d'objets dans chaque sous-ensemble.

Nous utilisons le réseau YOLOv3 du *framekwork* Darknet [39], implémentation officielle de ce réseau. Nous avons optimisé le réseau par descente de gradient stochastique avec les paramètres par défaut du code utilisé : des *batches* de 64 images, un taux d'apprentissage initial de 0,001 régulièrement divisé par 10 lorsque la précision sur l'ensemble de validation cessait d'augmenter, un *momentum* de 0,9, et une régularisation par *weight decay* de coefficient 0,0005.

4.2.4.2 Résultats sur les images synthétiques

Dans un premier temps, nous évaluons la capacité de YOLOv3 à reconnaître les objets d'intérêt dans des images synthétiques, après un entraînement avec des images synthétiques. Cela nous donne un aperçu de la capacité d'apprentissage du réseau, et constitue le cas idéal dans lequel les images de test ressemblent aux images d'entraînement, qui sont elles suffisamment nombreuses et variées. Les résultats sont rassemblés dans le Tableau 4.3, avec l'écart-type observé sur les trois sous-ensembles de la validation croisée.

Les résultats obtenus indiquent qu'en entraînant le réseau YOLOv3 de bout en bout, il est possible de reconnaître tous les objets avec une précision de 88,6 % en moyenne. La localisation est également très satisfaisante avec 76,9 % des boîtes englobantes prédites correctement localisées. En revanche, lorsque seul le détecteur est entraîné, la mAP n'atteint que 63,4 % et la localisation seulement 49,5 % ; cette performance semble indiquer que les caractéristiques apprises lors du pré-entraînement ne sont pas les plus adéquates pour les images synthétiques. Cette observation est en accord avec nos attentes, car les images synthétiques sont visuellement différentes des images réelles : en figeant les poids de l'extracteur de caractéristiques, le réseau doit apprendre à détecter des objets à partir des caractéristiques qui lui sont fournies, sans optimiser ces caractéristiques pour les images synthétiques. De plus, on observe une plus grande différence dans la performance de chaque classe, indiquant que certains objets sont plus faciles à reconnaître que d'autres. En conclusion, YOLOv3 est parfaitement capable d'identifier nos objets d'intérêt dans des images synthétiques, mais le domaine synthétique ou réel sur lequel est entraîné l'extrac-

39. https://github.com/AlexeyAB/darknet

teur de caractéristiques a une grande influence sur les résultats.

TABLEAU 4.3 – Performance de YOLOv3 entraîné avec des images synthétiques RUSPA, test sur images synthétiques. La précision moyenne pour chaque classe, puis la moyenne pour toutes les classes (mAP) sont données. L'IoU est calculée avec une superposition de 50 %. Toutes les valeurs sont en %.

	Poignée	**Dossier**	**Profil gauche**	**Profil droit**	**Siège**	**mAP**	**IoU**
Net_Synth	90,8	86,7	90,0	88,8	86,8	**88,6**±0,3	**76,9**±1,0
Det_Synth	86,5	48,6	69,5	58,3	53,4	63,4±5,0	49,5±2,4

4.2.4.3 Résultats sur les images réelles

Nous testons ensuite sur des images réelles les réseaux entraînés avec les images synthétiques. Nous comparons ces résultats avec la performance de YOLOv3 entraîné avec des images réelles. Ces résultats sont indiqués dans le Tableau 4.4.

Nous observons tout d'abord une forte diminution de la performance lorsque les réseaux entraînés sur images synthétiques sont appliqués à des images réelles, ce qui est dû à l'écart de domaine. Alors que sur les images de test synthétiques, c'est Net_Synth qui permettait d'obtenir la meilleure performance, c'est Det_Synth qui obtient de meilleurs résultats ici avec une réduction de mAP inférieure. Nous en concluons que les caractéristiques apprises sur ImageNet, correspondant à des images réelles, permettent au réseau d'être plus efficace sur les images réelles de RUSPA. Au contraire, Net_Synth semble avoir appris des caractéristiques correspondant aux images synthétiques, ce qui rend plus difficile la détection des objets dans les images réelles.

Lorsque YOLOv3 est entraîné sur des images réelles et testé sur d'autres images réelles, il permet d'obtenir les meilleures valeurs de mAP et d'IoU. En effet, le problème d'écart à la réalité ne se pose pas. En revanche, la performance ne dépasse pas 50 %, ce qui souligne la difficulté à apprendre avec une toute petite quantité d'images, qui sont de plus très peu variées et ne montrent pas les objets sous tous les angles. Dans ce cas, ne pas figer les poids de l'extracteur de caractéristiques donne les meilleurs résultats, le réseau apprenant des caractéristiques correspondant mieux à nos objets d'intérêt et à nos images. A nouveau, il y a de grandes différences entre les classes, confirmant la difficulté à identifier certains objets.

TABLEAU 4.4 – Performance de YOLOv3 sur RUSPA, test sur images réelles. La précision moyenne pour chaque classe, puis la moyenne pour toutes les classes (mAP) sont données. L'IoU est calculée avec une superposition de 50 %. Toutes les valeurs sont en %.

	Poignée	**Dossier**	**Profil gauche**	**Profil droit**	**Siège**	**mAP**	**IoU**
Net_Synth	12,1	30,0	12,8	1,1	39,1	19,0±8,4	32,1±9,0
Det_Synth	37,35	54,7	18,4	18,3	45,4	34,8±4,8	41,6±3,8
Net_Real	51,7	70,3	18,8	25,5	63,8	**46**±12,0	**44,4**±10,8
Det_Real	48,3	65,4	15,7	21,2	55,1	41,1±3,8	42,7±13,7

Suite à l'observation que les caractéristiques apprises lors du pré-entraînement avec ImageNet permettent au réseau de mieux généraliser aux images réelles, lorsqu'il est

entraîné avec des images synthétiques, nous avons poursuivi avec cette stratégie d'entraînement. Nous avons donc répété l'entraînement du détecteur en remplaçant 50 % des images d'entraînement par leurs versions avec du flou (de mouvement ou Gaussien), puis avec des ombres artificielles, et enfin avec soit du flou soit des ombres. Ces variations dans les caractéristiques des images synthétiques leur donnent un aspect plus similaires aux images réelles égocentriques. Cela permet d'évaluer si le réseau généralise mieux aux images réelles, sans avoir besoin d'augmenter sa taille ou d'utiliser une stratégie d'entraînement plus complexe. Les résultats sont présentés dans le Tableau 4.5.

Les résultats obtenus indiquent une meilleure localisation des boîtes englobantes suite à l'ajout des ombres pendant l'entraînement (Det_Shadows) ; par rapport à Det_Synth. Sans être réalistes, les ombres appliquées semblent permettre au réseau de mieux identifier les contours des objets. Ensuite, l'ajout du flou augmente fortement la précision et la localisation : la mAP passe de 34,8 % à 44 %, et l'IoU de 41,6 % à 51,3 %, entre Det_Synth et Det_Blur. Le flou paraît donc un élément crucial à synthétiser afin de mieux généraliser aux images réelles égocentriques. Enfin, si l'on utilise à la fois du flou et des ombres synthétiques, la mAP atteint 45,8 %, la meilleure performance pour un entraînement sans images réelles, et très proche de la valeur de 46 % obtenue par le réseau Net_Real. Ces résultats semblent indiquer qu'une performance équivalente peut être obtenue avec une faible quantité d'images réelles et avec des images synthétiques. Une meilleure localisation est obtenue en utilisant uniquement du flou, sans ombres artificielles, mais utiliser les deux permet une meilleure précision.

TABLEAU 4.5 – Performance de YOLOv3 avec différentes caractéristiques synthétiques pour la base RUSPA, test sur images réelles. Toutes les valeurs sont en %.

	Poignée	**Dossier**	**Profil gauche**	**Profil droit**	**Siège**	**mAP**	**IoU**
Det_Shadows	38,5	54,9	20,8	19,9	39,6	34,8±3,3	46,1±6,5
Det_Blur	50,8	52	28,9	29,6	61,5	44,0±1,4	**51,3**±5,0
Det_BS	47,2	59,4	26,9	32,9	62,2	**45,8**±1,5	46,5±3,7

Enfin, nous avons observé l'effet d'utiliser à la fois des images réelles et synthétiques pendant l'entraînement. Nous partons de l'entraînement de YOLOv3 avec des images synthétiques, et nous y intégrons les images réelles de deux façons : soit comme *fine-tuning* à la fin pour optimiser la tête de détection uniquement, soit en les mélangeant aux images synthétiques pendant l'entraînement. Les résultats sont dans le Tableau 4.6.

De façon générale, le *fine-tuning* avec les images réelles améliore systématiquement la performance : Net_Synth passe de 19 % à 40,8 % de mAP, Det_Synth de 34,8 % à 49,3 %, et Det_Blur de 44 % à 52,4 %. Bien que ces méthodes demandent plus de temps lors de l'entraînement, cette approche est plus efficace que le mélange des deux modalités pendant l'entraînement, qui ne permet d'atteindre que 48,5 % de mAP (ce qui est tout de même plus élevé que toutes les expériences sans images réelles).

Enfin, il est à noter que dans toutes les expériences que nous avons effectuées, il y a de grandes disparités entre les classes détectées. En particulier, les deux profils métalliques sont les objets les moins bien identifiés. Cela peut s'expliquer par le fait qu'ils apparaissent toujours très partiellement dans les images réelles, et qu'il est d'autant plus difficile de les reconnaître l'un de l'autre dans ces conditions. Pour l'annotation des images, nous nous sommes appuyés sur les séquences d'images successives afin d'annoter la bonne classe, car il n'était parfois pas possible d'identifier l'objet uniquement à partir d'une image seule.

C'est aussi parfois le cas pour les classes « Dossier » et « Siège », qui ont la même forme à l'exception de leur longueur, bien que le réseau ne semble pas avoir de difficulté à les identifier.

TABLEAU 4.6 – Performance de YOLOv3 entraîné avec une combinaison d'images synthétiques et réelles pour la base RUSPA, test sur images réelles. Toutes les valeurs sont en %.

	Poignée	**Dossier**	**Profil gauche**	**Profil droit**	**Siège**	**mAP**	**IoU**
Net_Synth+FT	41,3	67,0	10,1	27,8	57,4	40,8±13,6	44,7±7,6
Det_Synth+FT	58,6	72,4	21,8	23,9	69,8	49,3±10,6	42,7±2,9
Det_Blur+FT	61,6	74,4	19,8	38,4	67,8	**52,4**±10,3	**48,5**±7,7
Det_SynthReal	50,8	72,4	21,8	23,9	69,8	48,5±6,2	41,8±5,3

4.2.4.4 Analyse du type de flou

Les meilleurs résultats sont obtenus en appliquant du flou aux images d'entraînement. Plus précisément, nous avons appliqué soit du flou de mouvement, soit du flou Gaussien, choisi de façon aléatoire pour chaque image, à la moitié des 10 000 images d'entraînement synthétiques. Pour avoir une idée plus précise de l'influence de ces deux types de flou, nous avons répété cette expérience en n'utilisant que l'un ou l'autre. Ces résultats sont présentés sur la Figure 4.2.

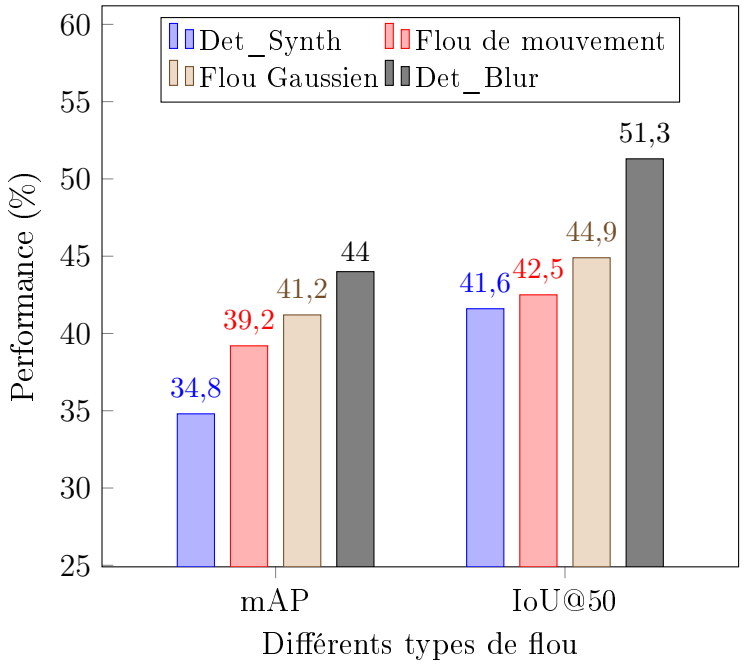


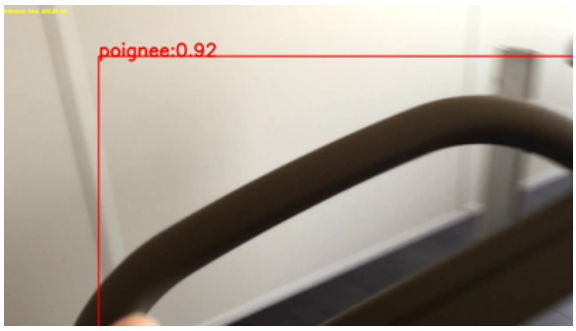
FIGURE 4.2 – Comparaison de la performance avec différents types de flou utilisés pendant l'entraînement de YOLOv3.

Les résultats obtenus indiquent que n'importe quel type de flou est mieux qu'aucun, étant donné que toutes les expériences permettent d'obtenir de meilleurs résultats que Det_Synth. Ensuite, utiliser du flou Gaussien seulement donne une meilleure performance sur les images réelles que du flou de mouvement seul. Nous pouvons supposer

que le flou de mouvement n'est pas présent dans toutes les images réelles, et il n'améliore donc la performance que sur une partie des images réelles. Au contraire, le flou Gaussien élimine le côté « trop net » des images synthétiques, ce qui réduit l'écart de domaine pour toutes les images. Enfin, utiliser les deux types de flou à parts égales pendant l'entraînement (Det_Blur) permet d'obtenir la meilleure performance, confirmant ainsi l'importance d'avoir des images d'entraînement variées.

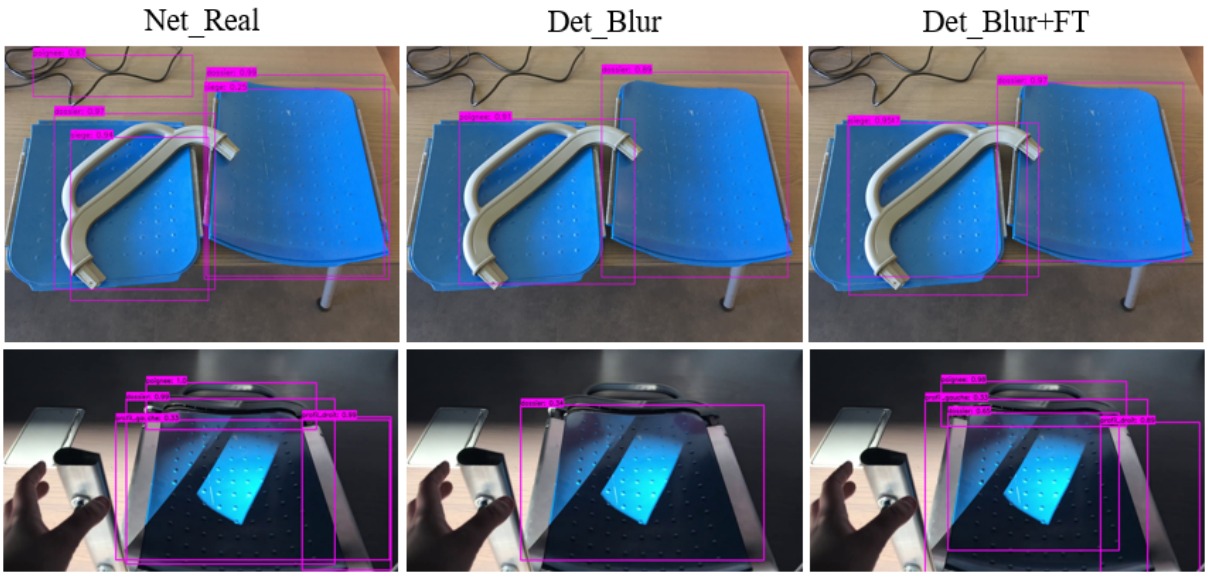
4.2.4.5 Analyse qualitative et discussion

Pour terminer l'analyse du détecteur d'objets YOLOv3, nous avons observé les détections obtenues par les différents réseaux sur quelques images réelles. Ces détections sont représentées sur la Figure 4.3.



(a) Objet partiellement visible (Det_Blur).

(b) Objet tronqué et flou (Det_Blur).



(c) Multiples objets par image.

FIGURE 4.3 – Visualisation des détections obtenues avec YOLOv3. Les images (a) et (b) n'appartiennent pas à l'ensemble de test mais sont extraites d'une séquence vidéo acquise ultérieurement ; elles indiquent la classe et l'indice de confiance (entre 0 et 1) du détecteur. Les images de (c) font partie de l'ensemble de test.

Sur la Figure 4.3a, on observe qu'un objet est correctement détecté, bien que partiellement visible, ce qui correspond à nos images d'entraînement synthétiques. Pour rappel, les objets peuvent apparaître hors de l'image jusqu'à un tiers de leur hauteur et largeur.

L'image de la Figure 4.3b est similaire, avec du flou causé par le mouvement de l'utilisateur, ces deux images étant extraites d'une séquence vidéo. A nouveau, bien que la boîte englobante ne contienne pas totalement l'objet, celui-ci est correctement détecté malgré le flou, et avec un indice de confiance de 100%. Pour ces deux images, c'est le réseau dénoté Det_Blur qui a obtenu ces résultats, c'est-à-dire le réseau entraîné sans aucune image réelle et avec un mélange de flous Gaussien et de mouvement.

Lorsque plusieurs objets sont superposés dans les images réelles, les détections obtenues diffèrent fortement en fonction de l'entraînement du réseau (Figure 4.3c). Si des images réelles ont été utilisées, que ce soit pour entraîner tout le réseau sans images synthétiques (Net_Real) ou comme *fine-tuning* (Det_Blur+FT), le réseau identifie un plus grand nombre de boîtes englobantes par image. Au contraire, lorsque le réseau n'est entraîné qu'avec des images synthétiques, il ne prédit pas de boîtes englobantes qui se superposent fortement, même si elles correspondent à des classes différentes. Cette observation est cohérente avec les images synthétiques que nous avons générées et utilisées ici, qui ne comportent qu'un objet par image. Le réseau n'a donc pas été confronté pendant l'entraînement à des boîtes englobantes ou des objets superposés. De plus, bien que les images réelles utilisées pendant l'entraînement proviennent de séquences vidéos différentes des images de test, le siège de bus complet apparaît probablement dans les deux ensembles d'images. En effet, les pièces sont toujours dans les mêmes positions relatives les unes par rapport aux autres lorsque le siège est assemblé. Il est possible que cette information ait été apprise par le réseau dans les images réelles, ce qui facilite la détection dans les images de test. Dans les images montrées ici, il semble que cela aide en particulier le réseau à identifier les objets très similaires entre eux, comme le dossier et l'assise du siège de bus (le dossier est la pièce liée à la poignée, alors que l'assise non), ou les profils droit et gauche lorsqu'ils sont partiellement visibles (qui sont plus facilement identifiés par rapport à la pièce centrale que seuls).

Bien que les images réelles améliorent la détection, elles introduisent aussi des erreurs qui se traduisent par une trop grande quantité de boîtes englobantes. En effet, l'image supérieure gauche de la Figure 4.3c montre une détection du câble posé sur la table comme la poignée (qui elle, n'est pas détectée), ainsi qu'une double détection du dossier et du siège, à la fois à l'emplacement du dossier et du siège. Dans l'image supérieure droite de la même figure, qui correspond à un entraînement avec des images sythétiques puis du *fine-tuning* avec des images réelles (Det_Blur+FT), une partie de ces détections parasites a disparu, mettant en valeur la complémentarité des images synthétiques et réelles.

4.3 Détection avec SSD et des images RGB issues de scènes 3D

Pour la suite de nos travaux, nous avons remplacé le réseau YOLOv3 par SSD Lite avec MobileNetV3 comme extracteur de caractéristiques, tel qu'il est décrit dans la publication Howard *et al.* (2019) et dans la section 2.3.2. Nous avons été motivés pour changer d'architecture du fait de la rigidité du *framework* Darknet, qui rend le réseau difficilement utilisable si l'on souhaite modifier l'entraînement ou l'architecture. Avec une implémentation dans le *framework* PyTorch, il est plus simple de modifier l'architecture du réseau considéré, par exemple pour ajouter une modalité en entrée (voir Chapitre 5) [40]. De plus,

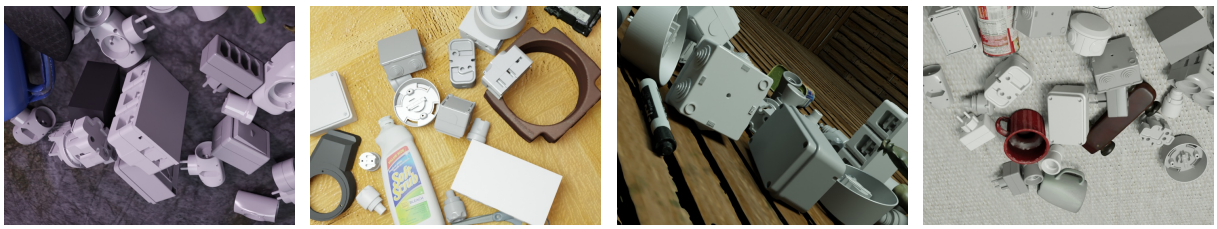
40. Les successeurs de YOLOv3, disponibles dans le *framework* PyTorch, n'ont été publiés qu'après le début de nos travaux.

le réseau SSD offre plus de flexibilité quant à son extracteur de caractéristiques : il est facilement interchangeable, ce qui nous permet de comparer différentes tailles de réseau.

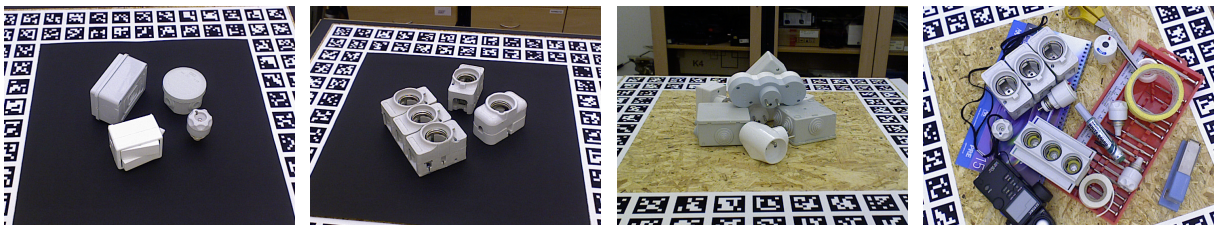
Notre premier objectif est d'identifier si la plus petite architecture composée de MobileNetV3 *Small* donne une performance suffisante lorsqu'elle est entraînée sur des images synthétiques seulement, ou si une architecture plus conséquente est nécessaire avec MobileNetV3 *Large*. En effet, nous voulons la plus petite architecture possible, mais pas au prix d'une performance insuffisante.

Afin de valider la pertinence de cette architecture pour l'apprentissage sans aucune image réelle, nous l'entraînons avec les images RGB synthétiques et photo-réalistes du jeu de données T-LESS (Hodan *et al.*, 2017), une base publique utilisée pour le benchmark d'algorithmes d'estimation de pose, et dont les objets ont une apparence industrielle (Figure 4.4). Ces images, décrites plus en détail dans le paragraphe 5.3.1, sont générées en construisant une scène 3D comprenant les objets, et en capturant des images grâce une caméra virtuelle avec l'outil BlenderProc (présenté dans le paragraphe 3.3.1). Nous avons choisi ce jeu de données à la place de notre cas d'usage RUSPA afin de réduire la difficulté de la tâche : si l'architecture SSD avec MobileNetV3 est capable de reconnaître les objets de T-LESS dans les images réelles en apprenant avec des images totalement synthétiques, alors nous pouvons dans un second temps ajouter les difficultés liées au point de vue égocentrique.

Nous en profitons également pour explorer la phase d'augmentation des données, notre deuxième objectif. En effet, nous avons observé que les images synthétiques générées à partir de scènes 3D ont une apparence moins réaliste que les images 2D augmentées, en ce qui concerne le bruit et la netteté de l'image, ainsi que l'arrière-plan. Nous évaluons donc également l'influence de l'augmentation de données « forte » que nous proposons dans la partie 4.3.2, en la comparant avec l'augmentation de données standard.



(a) Images RGB synthétiques.



(b) Images RGB réelles.

FIGURE 4.4 – Exemples d'images RGB du jeu de données T-LESS.

4.3.1 Architecture de SSD avec MobileNetV3 *Large* et *Small*

Nous étudions la performance de MobileNetV3 SSD afin de choisir entre les extracteurs de caractéristiques *Large* et *Small* de la famille MobileNetV3. Cette architecture est représentée sur la Figure 4.5. Comme décrit dans le paragraphe 2.3.2, le réseau SSD

est composé d'un extracteur de caractéristiques, qui peut être n'importe quel classifieur auquel on a enlevé ses couches de prédictions, de 4 couches *extra* obtenues par convolution successives qui créent des caractéristiques à plusieurs échelles, et de 6 couches de prédiction, connectées aux couches *extra* et à deux couches de caractéristiques intermédiaires de MobileNetV3 (dénotées C4 et C5).

Tout comme YOLO, les prédictions de SSD sont associées à des boîtes prédéfinies de différentes tailles pour chacune des 6 couches de caractéristiques définies. Pour chaque boîte prédéfinie, le réseau produit à la fois des indices de confiance pour chaque classe, correspondant à la probabilité qu'un objet de cette classe soit dans la boîte, et des coordonnées ajustées à l'objet afin d'affiner la localisation. Le nombre de boîtes prédéfinies peut être choisi, et dépend de la définition spatiale de l'image d'entrée et des couches de caractéristiques. Dans notre cas, nous obtenons 1602 prédictions, qui sont triées par une étape de suppression des non-maximums (NMS). Cet algorithme élimine d'abord les prédictions correspondant à l'arrière-plan, puis celles d'indice de confiance inférieur à 10%, et enfin celles de même classe qui se superposent de plus de 50%, en gardant celle d'indice de confiance le plus élevé. Les prédictions restantes sont ensuite évaluées par rapport aux objets de la vérité terrain.

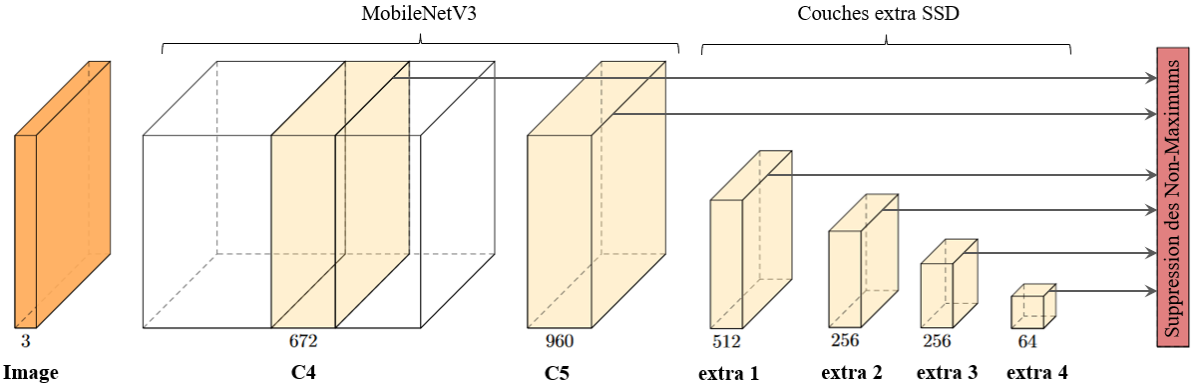


FIGURE 4.5 – Architecture du détecteur d'objets SSD avec MobileNetV3 *Large* comme extracteur de caractéristiques.

4.3.2 Augmentation de données forte

Nous entraînons cette architecture avec des images synthétiques générées à partir de scènes 3D. Pour réduire l'écart de domaine avec les images réelles, une augmentation plus prononcée est nécessaire. En effet, les images synthétiques générées par rendu de scènes 3D sont nettes, sans défaut, alors que les images réelles sont généralement plus bruitées. En particulier, si les images sont acquises par des caméras standard du commerce, ou avec du mouvement, elles présentent une moins bonne qualité et des artefacts auxquels le réseau n'aura pas été confronté pendant l'entraînement. Nous présentons donc dans cette partie notre approche pour augmenter les images RGB afin de réduire l'écart à la réalité. Cette procédure d'augmentation est appliquée à chaque image synthétique avant de traverser le réseau de neurones, avec de nouveaux paramètres choisis aléatoirement permettant d'assurer que le réseau ne verra pas plusieurs fois exactement la même image.

Pour réduire cet écart de domaine dans les images couleurs, nous avons proposé une séquence d'opérations d'augmentation plus complexe que la procédure standard, et dont les

paramètres prennent des valeurs dans des intervalles élargis par rapport aux travaux précédents de la littérature. Les augmentations standard sont présentées dans le Tableau 4.7, et notre augmentation « forte » dans le Tableau 4.8. Ces tableaux comprennent à la fois les augmentations dans l'espace colorimétrique et les déformations géométriques (*VerticalFlip* et *RandomCrop*), chaque ligne du tableau étant appliquée de façon séquentielle avec une certaine probabilité pour une image donnée. Les augmentations standards présentées ici sont celles utilisées pour entraîner le réseau CosyPose (Labbé *et al.*, 2020), état de l'art pour l'estimation de pose d'objets à partir d'images synthétiques, avec lequel nous comparons nos résultats.

Dans la continuité de nos premiers résultats avec YOLOv3 et les images 2D augmentées, nous avons intégré une opération appliquant du flou, avec une probabilité de 50 %, dont les paramètres sont choisis aléatoirement. En ce qui concerne les ombres artificielles, elles sont déjà présentes dans les images générées à partir de scènes 3D, car ces scènes comportent plusieurs sources lumineuses et plusieurs objets, dont les positions relatives créent des ombres variées et réalistes. Nous n'avons donc pas appliqué ici les ombres artificielles comme sur les images 2D augmentées.

TABLEAU 4.7 – Séquences d'opérations pour une augmentation de données standard, avec des fonctions de la bibliothèque logicielle PIL [41].

Opérations et valeurs de paramètres	**Probabilité**
GaussianBlur(factor_interval=[1, 3])	0,4
Sharpness(factor_interval=[0, 50])	0,3
Contrast(factor_interval=[0,2, 50])	0,3
Brightness(factor_interval=[0,1, 6])	0,5
Color(factor_interval=[0, 20])	0,3

4.3.3 Expériences et résultats

Dans cette partie, nous décrivons les différentes expériences menées avec le réseau SSD et le jeu de données T-LESS. Précisément, nous avons entraîné le réseau SSD avec MobileNetV3 *Large* et MobileNetV3 *Small* comme extracteur de caractéristiques, et en appliquant les deux méthodes d'augmentation de données. Nous comparons également les résultats obtenus avec un réseau de la littérature entraîné sur les mêmes images synthétiques, mais ayant une architecture beaucoup plus conséquente, Mask R-CNN (He *et al.*, 2020).

4.3.3.1 Détails d'entraînement

Tous les réseaux sont pré-entraînés sur le jeu de données ImageNet. L'implémentation du réseau MobileNetV3 SSD utilisée ici n'est pas une implémentation officielle, mais un projet open source [43] disponible sur GitHub et utilisant le *framework* PyTorch. Nous avons rendu disponible en ligne le code de test correspondant à nos expériences [44].

Les réseaux ont été optimisés de bout en bout par descente de gradient stochastique

41. `https://pillow.readthedocs.io/en/stable/`
42. `https://albumentations.ai/docs/`
43. https://github.com/d-li14/mobilenetv3.pytorch
44. `https://gitlab.liris.cnrs.fr/jcohen/synthetic-ssd`

TABLEAU 4.8 – Séquences d'opérations pour notre augmentation de données « forte », avec des fonctions de la bibliothèque logicielle *albumentations*[42] (Buslaev *et al.*, 2020).

Opérations et valeurs de paramètres	Probabilité
Choisir une opération parmi	
ColorJitter(brightness=[0,5, 1,4], contrast=0,5, saturation=0,9, hue=0,5) ColorJitter(brightness=0, contrast=0, saturation=0, hue=0,5) ColorJitter(brightness=0, contrast=0, saturation=0,9, hue=0) ColorJitter(brightness=0, contrast=0,5, saturation=0, hue=0) ColorJitter(brightness=[0,5, 1,4], contrast=0, saturation=0, hue=0)	0,8
CLAHE()	0,5
RGBShift()	0,5
Choisir une opération parmi	
Blur(blur_limit=[3, 7]) GaussianBlur(blur_limit=[3, 7], sigma_limit=0) MedianBlur(blur_limit=[3, 7]) MotionBlur(blur_limit=[5, 15])	0,5
GaussNoise()	0,8
MultiplicativeNoise(multiplier=[0,7, 1,3])	0,2
ISONoise()	0,2
VerticalFlip()	0,5
RandomCrop(height=400, width=400)	0,6

avec les paramètres suivants : taux d'apprentissage de $5e^{-2}$, *momentum* de $9e^{-1}$, *weight decay* de $12e^{-6}$, et une taille de *batch* de 32 images. Ces hyperparamètres ont été déterminés expérimentalement par recherche aléatoire (*random search*) avec le réseau MobileNetV3 Large, et appliqués à toutes les architectures comparées. Tous les réseaux ont été pré-entraînés avec le jeu de données ImageNet.

Nous comparons la performance obtenue avec celle d'un réseau Mask R-CNN entraîné sur les mêmes images synthétiques. Dans la publication de Labbé *et al.* (2020), un réseau RetinaNet est entraîné avec des images synthétiques afin d'extraire chaque objet, comme première étape avant d'en estimer la position dans l'espace. Ce réseau est utilisé pour permettre une comparaison équitable avec les autres approches de l'état de l'art, mais n'est pas celui qui offre la meilleure performance. Pour cela, les auteurs ont remplacé RetinaNet par un réseau Mask R-CNN, dont les résultats et poids entraînés sont disponibles en ligne[45]. Nous l'avons appliqué directement sur les images de test, de la même façon que nos réseaux MobileNetV3 SSD *Large* et *Small*.

4.3.3.2 Résultats

Le Tableau 4.9 indique la performance obtenue par les différents réseaux sur le jeu de données T-LESS, ainsi que leurs tailles respectives. Le réseau Mask R-CNN obtient une précision moyenne de 32,8 %, les images d'entraînement étant augmentées avec la méthode standard. Comme attendu, le modèle MobileNetV3 *Small* obtient une performance inférieure à celle de MobileNetV3 *Large*. Pour ces deux architectures, l'augmentation de données forte permet une importante amélioration de la performance : de 18,6 % à 23,5 % pour le réseau *Small*, et de 36,3 % à 46,1 % pour le réseau *Large*. Avec les deux niveaux d'augmentation, le réseau MobileNetV3 SSD *Large* obtient une meilleure performance

45. `https://github.com/ylabbe/cosypose`

que Mask R-CNN, malgré sa taille bien inférieure. Nous supposons que cette faible performance de Mask R-CNN est dûe à la définition basse de nos images en entrée du réseau : nous réduisons les images à une définition de 224×224 pixels (ce qui correspond à SSD), alors que Mask R-CNN est entraîné avec des images bien plus grandes. Augmenter la taille des images augmenterait probablement la performance de Mask R-CNN, mais augmenterait également sa durée d'inférence, déjà bien plus longue que SSD. En ce qui concerne la taille du réseau, Mask R-CNN contient 44 millions de paramètres, contre moins de 5 millions de paramètres pour les différentes versions de MobileNetV3 SSD. Ces résultats nous encouragent à utiliser le réseau MobileNetV3 SSD *Large* pour la suite des expériences.

TABLEAU 4.9 – Performance de SSD sur les images réelles du jeu de données T-LESS.

Architecture	**Augmentation**	**mAP (%)**	**Paramètres (M)**
Mask R-CNN	standard	32,8	44,0
MobileNetV3 SSD *Small*	standard	18,6	2,6
MobileNetV3 SSD *Large*	standard	36,3	4,9
MobileNetV3 SSD *Small*	forte	23,5	2,6
MobileNetV3 SSD *Large*	forte	**46,1**	4,9

4.4 Conclusion

Nous avons étudié deux architecture de réseaux de neurones de type *single-stage* pour la détection d'objets dans des images RGB. Ces deux réseaux, YOLOv3 et SSD Lite, permettent une détection suffisamment rapide pour un traitement des images en temps réel. Dans une application industrielle, l'implémentation de ces réseaux sera également optimisée, en taille occupée en mémoire ainsi qu'en durée d'inférence.

Le réseau YOLOv3 nous a permis de valider l'approche qui consiste à utiliser des images synthétiques composées de modèles 3D projetés sur une images réelle. En particulier, l'étude des caractéristiques égocentriques de ces images synthétiques nous a permis d'identifier des éléments cruciaux pour une bonne généralisation aux images réelles, tel que la présence de flou. Une étude plus approfondie pourrait être menée, faisant varier le nombre d'images synthétiques ainsi que le nombre d'images comportant du flou et des ombres, afin d'identifier la combinaison optimale pour ce jeu de données et ce réseau. L'utilisation d'images réelles pendant l'entraînement nous a permis de mettre en lumière quelques éléments manquants à nos images synthétiques. En premier lieu, la présence de plusieurs objets dans les images synthétiques permettrait de mieux détecter les superpositions et occultations dans les images réelles. Ensuite, les détections obtenues semblent indiquer que le réseau entraîné avec des images réelles a appris à reconnaître les objets par rapport à leurs positions relatives. Cette intuition, qui semble soulever un biais dans les données utilisées ici, peut également être vue comme une piste d'amélioration pour le générateur d'images synthétiques. En effet, nous avons considéré jusque-là les pièces séparément, mais il serait aussi possible de générer des images synthétiques montrant les pièces assemblées, totalement ou partiellement.

Ces images réelles nous ont également permis de montrer que le réseau YOLOv3 est capable d'extraire et combiner de l'information à partir des images des deux domaines. Il serait intéressant de quantifier l'amélioration obtenue grâce aux images réelles au-delà de leur simple absence ou présence, notamment en diminuant leur nombre à seulement

quelques images par objet. De plus, nous avons obtenu les meilleurs résultats en gardant l'extracteur de caractéristiques pré-entraîné sur ImageNet, sans optimisation supplémentaire. Des travaux ultérieurs pourraient étudier une optimisation différente de ces poids, par exemple en entraînant le réseau de bout-en-bout avec les images réelles avant d'utiliser les images synthétiques. Bien que peu conventionnelle, cette approche permettrait d'orienter les caractéristiques extraites des images vers le domaine réel, avant d'apprendre au réseau à reconnaître ces objets grâce aux images synthétiques.

Après avoir montré qu'il était possible d'atteindre une performance aussi bonne avec des images synthétiques qu'avec des images réelles, nous cherchons à la surpasser pour nous passer complètement des images réelles pendant l'entraînement. Inspirés par les travaux du domaine de la réalité augmentée, nous nous sommes tournés du côté des images RGB-D afin d'apporter un supplément d'information aux images couleur. Cependant, nous avons observé que le réseau YOLOv3 ainsi que ces images synthétiques ne permettent pas facilement l'ajout d'une modalité supplémentaire, comme une carte de profondeur. Par conséquent, nous avons remplacé YOLOv3 par SSD, et les images 2D augmentées par des images issues de scènes 3D entièrement synthétiques générées avec BlenderProc. Nous avons validé l'utilisation d'une nouvelle séquence d'opérations pour l'augmentation des images RGB, afin de réduire de façon plus efficace l'écart de domaine entre les images synthétiques et réelles. Bien que cette séquence d'augmentations ait été définie de manière empirique et doit pouvoir être encore affinée, elle atteint notre objectif d'amélioration de la précision sur les images réelles. Nous avons obtenu la meilleure performance avec le réseau MobileNetV3 *Large* comme extracteur de caractéristiques, que nous privilégions donc par rapport à MobileNetV3 *Small*.

Face aux nombreuses architectures de réseaux compacts, destinées aux applications mobiles, il serait intéressant de compléter ces analyses en remplaçant MobileNetV3 par d'autres extracteurs de caractéristiques. Cependant, de nouveaux réseaux sont proposés dans la littérature très régulièrement, toujours plus performants sur les bases de données de référence. Identifier la *meilleure* architecture est donc une quête permanente, qui demande du temps et beaucoup de ressources de calculs afin d'évaluer différents hyperparamètres et stratégies d'entraînement. Nous avons choisi de garder MobileNetV3 avec SSD Lite pour continuer nos travaux, un réseau qui a fait ses preuves dans de nombreuses applications embarquées, tant dans la communauté scientifique que dans l'industrie. Cela nous permet de nous concentrer sur l'amélioration de la détection d'objets avec une modalité supplémentaire, en nous basant sur une architecture maîtrisée et éprouvée.

Les travaux présentés dans ce chapitre ont fait l'objet de deux publications internationales. Notre méthode de génération d'images synthétiques égocentriques, ainsi que leur utilisation pour entraîner YOLOv3, ont été présenté à l'oral lors de la conférence VISAPP en février 2020 à Malte (Cohen *et al.*, 2020). L'article associé a reçu le prix du « Meilleur article industriel » de la conférence. En 2021, nous avons présenté les résultats obtenus avec le réseau SSD et le jeu de données T-LESS avec notre méthode d'augmentation « forte » lors d'une session poster virtuelle pendant la conférence ICIP (Cohen *et al.*, 2021a). Ces mêmes résultats ont également été présentés à l'oral et en français lors de la conférence ORASIS, également en septembre 2021 (Cohen *et al.*, 2021b).

Chapitre 5

Détection d'objets multimodale (RGB-D)

Ce chapitre présente l'extension des travaux du chapitre précédent à une seconde modalité : une carte de profondeur. Après avoir présenté l'approche que nous proposons pour combiner une image couleur et sa carte de profondeur associée dans un réseau de neurones qui reste compact et efficace, nous détaillons les différentes expériences menées afin d'améliorer la performance sans augmenter la taille du détecteur d'objets. En particulier, nous nous sommes concentrés sur la réduction de l'écart de domaine pour l'image de profondeur, dont la performance laissait à désirer. Nous utilisons une méthode par traitement d'images classique sur la carte de profondeur réelle, ce qui ne requiert pas d'apprentissage et améliore grandement les performances en réduisant la différence avec les images synthétiques. Ainsi, notre réseau multimodal est suffisamment performant pour surpasser les résultats d'un détecteur d'objets basé uniquement sur l'image couleur, malgré l'écart à la réalité.

Sommaire

5.1 Introduction

Les résultats obtenus dans le chapitre précédent ont pointé la difficulté d'un réseau de neurones compact à correctement identifier des objets dans des images réelles égocentriques lorsqu'ils sont entraînés uniquement avec des images synthétiques. En particulier, la performance atteint à peine 50% avec le jeu de données RUSPA. Etant donné que nous avons des modèles 3D à disposition, et que les appareils de réalité augmentée disposent généralement de plusieurs caméras, ce qui permet de reconstruire une carte de profondeur, nous nous sommes intéressés aux images RGB-D. En effet, les modèles 3D contiennent *a priori* plus d'informations que ce qui est projeté sur une image RGB, et cette information pourrait être utile à un réseau de neurones pour mieux localiser et identifier les objets.

Nous étudions donc dans ce chapitre l'utilisation de cartes de profondeur en association avec les images RGB. Cette approche, détaillée dans la section 5.2.2 et dont les résultats sont présentés à partir de la section 5.4.2, a demandé la création de nouvelles images annotées RGB-D, réelles et synthétiques, pour le siège de bus RUSPA ; c'est ce qui a motivé la création du jeu de données synthétique décrit dans la partie 3.3. Pour les images réelles, nous avons acquis des images RGB-D avec une caméra Intel RealSense D435i, comme décrit dans la section 5.3.1.

Après avoir réalisé une analyse préliminaire de la performance de SSD avec chacune des modalités séparément, nous avons proposé une nouvelle architecture multimodale afin de fusionner les deux. Nous avons fait un compromis entre une taille de réseau raisonnable, avec le moins de calculs possible ajoutés par rapport à la version uni-modale, et une performance satisfaisante pour les différents jeux de données comparés. En effet, il est important que la solution proposée convienne pour la détection d'objets correspondant à des objets différents, puisqu'elle a vocation à intégrer un processus industriel automatisé, avec le moins d'intervention humaine possible.

Lors de l'entraînement d'un réseau de neurones, de nombreux éléments jouent sur la performance finale : l'architecture, les paramètres d'entraînement, ainsi que les données d'entraînement et de test. Un même modèle appliqué à différentes bases de données ne donnera pas les mêmes résultats, notamment lorsque le domaine d'entraînement est très différent du domaine de test. Avec toujours comme objectif d'identifier les objets RUSPA dans des images égocentriques, nous avons également comparé les résultats obtenus avec deux jeux de données publics, LINEMOD et T-LESS, de difficultés variées. Ainsi, nous pouvons évaluer les points forts et les points faibles de notre approche en fonction des caractéristiques visuelles des images considérées.

5.2 Architecture pour la détection d'objets RGB-D

5.2.1 MobileNetV3 SSD Lite

Nous utilisons comme détecteur d'objets de référence le réseau SSD Lite avec MobileNetV3 *Large* comme extracteur de caractéristiques, tel qu'il est proposé dans l'article introduisant MobileNetV3 (Howard *et al.*, 2019). Il s'agit du même réseau que celui utilisé dans le chapitre précédent (partie 4.3). Nous l'entraînons avec les images de chacune des modalités séparément, afin de connaître la performance de référence pour chaque jeu de données. A partir de ce réseau, nous définissons une architecture multimodale qui sera entraînée avec les deux modalités en même temps.

5.2.2 SSD multimodal

Nous décrivons ici l'architecture multimodale que nous avons proposée pour faire de la détection d'objets dans des images RGB-D (Figure 5.1). L'objectif est de créer un modèle qui n'implique que très peu de calculs supplémentaires et qui occupe une place en mémoire limitée. L'architecture proposée est basée sur le détecteur SSD, mais elle peut s'appliquer à d'autres architectures de détection d'objets *single stage*.

Le principe est le suivant : nous remplaçons l'extracteur de caractéristiques par une paire d'extracteurs de caractéristiques, chacun spécifique à une modalité. Nous adoptons une approche par fusion intermédiaire, en fusionnant des couches de caractéristiques de chaque extracteur, afin de fournir au détecteur d'objets des caractéristiques unifiées. Ainsi, la partie du détecteur d'objets chargée de prédire les localisations des boîtes englobantes et la classe des objets n'est pas structurellement modifiée (bien qu'un changement de taille des couches soit possible). Contrairement à de précédents travaux (Xu *et al.*, 2017; Fan *et al.*, 2021), nous ne créons pas de branche d'extraction de caractéristiques jointes entre les modalités. En effet, cette branche supplémentaire ajoute un nombre conséquent de neurones et d'opérations, et nous émettons l'hypothèse que les deux modalités s'influencent déjà entre elles au travers de l'optimisation du réseau de bout en bout. Ainsi, nous fusionnons les caractéristiques des deux modalités à deux niveaux avec un seul réseau SSD, contrairement à l'architecture de Kim *et al.* (2018), qui comprend un réseau SSD par modalité et dans laquelle 4 couches *extra* sont également fusionnées.

Plus formellement, soit x_m une image RGB-D, avec $(m \in \{RGB, P\})$ tel que x_{RGB} est la composante RGB et x_P est la composante de profondeur. On note Φ_4 et Φ_5 les opérations de fusion correspondant aux cartes de caractéristiques C4 et C5, respectivement. Les caractéristiques fusionnées $C4_{fused}$ et $C5_{fused}$ sont obtenues après le passage des images dans chacun des extracteurs de caractéristiques, puis à travers les couches de fusion, comme indiqué par les équations suivantes :

$$C4_{fused} = \Phi_4(C4_{RGB}, C4_P) \tag{5.1}$$

$$C5_{fused} = \Phi_5(C5_{RGB}, C5_P) \tag{5.2}$$

Les couches de prédiction de SSD sont connectées à $C4_{fused}$ et $C5_{fused}$, pour effectuer le parallèle avec les couches C4 et C5 du cas unimodal ; les couches *extra* sont connectées à $C5_{fused}$. Les prédictions sont effectuées à partir des deux couches fusionnées, ainsi que des quatre couches *extra*, comme pour le réseau SSD classique. Ces deux couches de caractéristiques fusionnées, issues des extracteurs de caractéristiques, présentent des résolutions spatiales différentes, ainsi que des caractéristiques de plus ou moins haut niveau sémantique, apportant ainsi des informations complémentaires pour la phase de prédiction.

Les poids des deux extracteurs de caractéristiques MobileNetV3 sont indépendants. La fonction de perte à optimiser est la même que pour le réseau SSD à une seule modalité, la fonction MultiBox (Liu *et al.*, 2016). Cet objectif comprend une fonction de perte liée à la localisation de l'objet dans une image, dénotée L_{loc}, et une autre liée à la classification de l'objet dans chaque boîte englobante identifiée, dénotée L_{conf}. Pour une paire d'images x_m en entrée, et des boîtes englobantes localisées par les coordonnées l et associées aux scores de confiance c, la fonction de coût totale est calculée de la façon suivante :

$$L(x_m, l, c) = L_{loc}(x_m, l) + L_{conf}(x_m, c) \tag{5.3}$$

Le détail des fonctions de perte partielles L_{loc} et L_{conf} est donné dans les équations 2.3.2.3 et 2.3.2.3 respectivement.

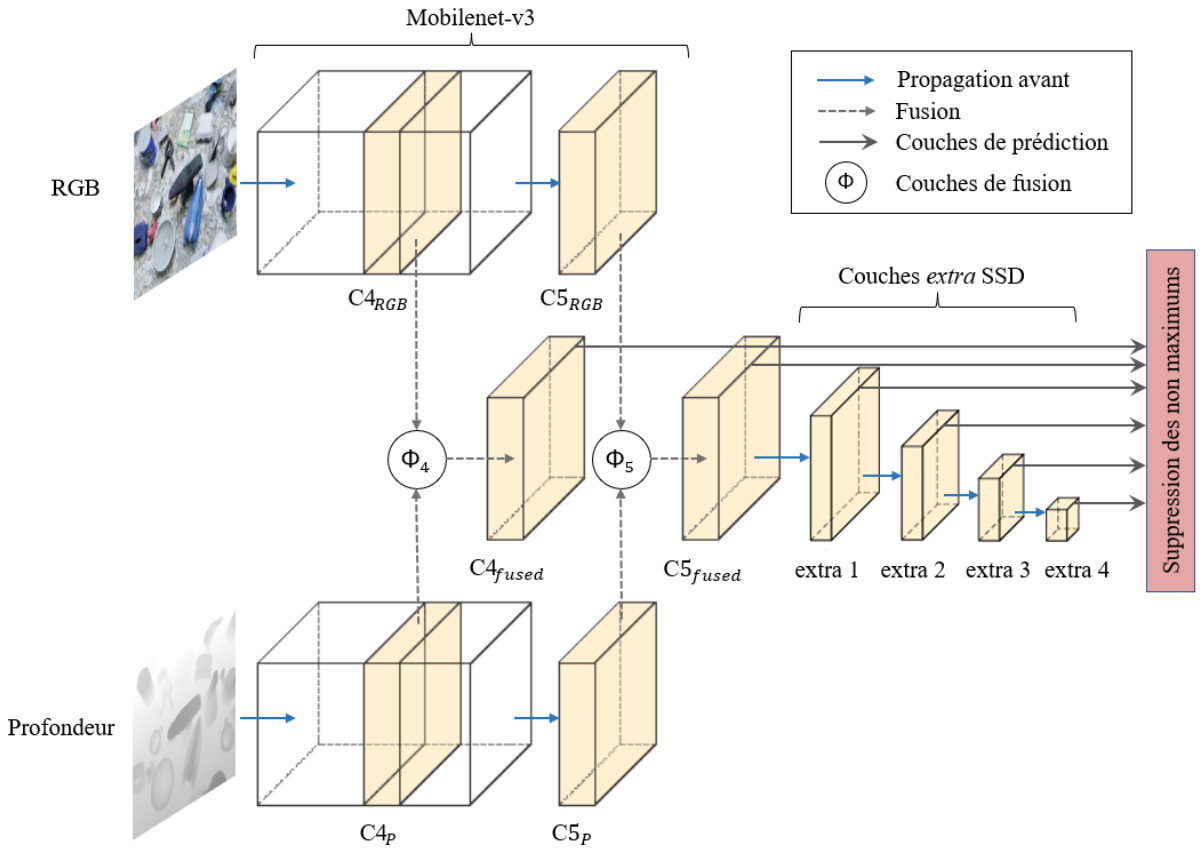


FIGURE 5.1 – Architecture SSD multimodale proposée pour la détection d'objets dans des images RGB-D : deux extracteurs de caractéristiques MobileNetV3 fournissent des caractéristiques issues des couches C4 et C5 pour chaque modalité ; deux couches de fusion ϕ combinent ces caractéristiques avant de les fournir au réseau SSD.

5.2.2.1 Méthodes de fusion de modalités

Plusieurs méthodes de fusion simples de cartes de caractéristiques ont été proposées dans la littérature, que nous avons comparées. Par la suite, on note c_m^i l'élément d'indice i d'une carte de caractéristiques c_m qui sera fusionnée ; c_{RGB}^i et c_P^i sont donc des éléments de même position spatiale, correspondant à nos deux modalités RGB et profondeur ; c_{fused}^i correspond à l'élément d'indice i de la carte de caractéristiques produite par l'opération de fusion.

a) Fusion par concaténation La méthode de fusion de modalités la plus naïve est la simple concaténation des couches de caractéristiques, selon la dimension de la profondeur (Figure 5.2a). Cette approche ne requiert pas de paramètre supplémentaire pour le réseau, en revanche les caractéristiques obtenues en sortie sont plus profondes que les caractéristiques de chaque modalité. Par conséquent, le détecteur d'objets qui reçoit ces caractéristiques en entrée doit être adapté pour que le nombre de filtres de convolution coïncide.

$$c_{fused} = c_{RGB}^i \oplus c_P^i \tag{5.4}$$

b) Fusion par moyenne Lorsque les cartes de caractéristiques sont de mêmes dimensions, la fusion peut s'opérer en appliquant une opération entre les éléments des deux

modalités. Moyenner les cartes de caractéristiques élément par élément (Figure 5.2b) est une méthode de fusion proposée dans Hoffman *et al.* (2016) qui a montré de meilleurs résultats que la simple concaténation, dans leur cas. Cela permet de prendre en compte les deux modalités de façon équivalente. En revanche, cela peut aussi perdre l'information des deux modalités si les éléments activés ne correspondent pas du tout aux mêmes zones de l'image. Cette méthode ne demande à nouveau pas de paramètre supplémentaire à apprendre, et a l'avantage de produire des caractéristiques de même taille que celles de chaque modalité en entrée. Pour le détecteur d'objets, cela revient à la même architecture que s'il n'y avait qu'une seule modalité.

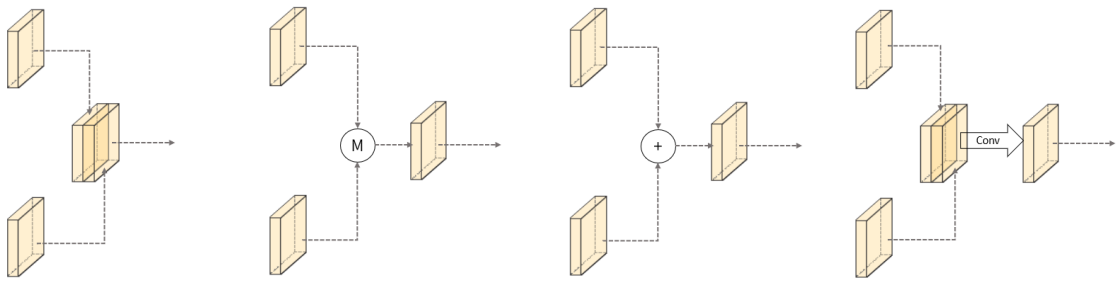
$$c^i_{fused} = \frac{c^i_{RGB} + c^i_P}{2} \tag{5.5}$$

c) Fusion par addition Pour éviter la limitation de la fusion par moyenne, il est possible d'additionner les cartes de caractéristiques, élément par élément (Figure 5.2c). Cela permet de faire ressortir les valeurs les plus élevées pour les deux modalités, et ainsi d'éviter la perte d'informations. Comme pour la moyenne, la taille du détecteur est inchangée et il n'y a aucun paramètre supplémentaire.

$$c^i_{fused} = c^i_{RGB} + c^i_P \tag{5.6}$$

d) Fusion par convolution L'un des avantages des réseaux de neurones par rapport à d'autres algorithmes est leur capacité à extraire l'information nécessaire à partir des données fournies. Plutôt que d'indiquer au réseau la meilleure façon de combiner les modalités, nous proposons donc de le laisser apprendre cette combinaison avec une couche de convolution apprise durant l'entraînement, de paramètres W et de biais b (Figure 5.2d). Les cartes de caractéristiques de chaque modalité sont d'abord concaténées, puis passées à travers un bloc de convolution, avant d'être fournies au détecteur d'objets. Un avantage de cette méthode de fusion est que la taille des caractéristiques en sortie peut être choisie avec les paramètres de la couche de convolution. Nous fixons les dimensions de ces caractéristiques de sortie identiques à celles des couches d'entrée, pour ne pas influer sur la taille du détecteur d'objets. Cette méthode demande toutefois des calculs supplémentaires dûs à la couche de convolution insérée.

$$c^i_{fused} = (c^i_{RGB} \oplus c^i_P) * W + b \tag{5.7}$$



(a) Concaténation (b) Moyenne (c) Addition (d) Convolution

FIGURE 5.2 – Illustration des différentes méthodes de fusion de cartes de caractéristiques.

5.2.2.2 Application à MobileNetV3 SSD

Une particularité de l'architecture SSD est qu'elle émet des prédictions à partir de deux endroits dans l'architecture de son extracteur de caractéristiques, et non uniquement en sortie. Ces deux sources sont notées C4 et C5 dans la littérature et sont représentées sur la Figure 5.1. Nous définissons donc l'architecture multimodale SSD avec deux modèles MobileNetV3, fusionnés au niveau des couches C4 et C5, et SSD utilise ces deux couches fusionnées pour détecter les objets d'intérêt.

5.3 Description des données d'entraînement et de test

5.3.1 Les jeux de données

Dans cette partie, nous présentons les différentes bases de données utilisées, en mettant l'accent sur leurs caractéristiques et particularités. Le nombre d'images d'entraînement et de test est indiqué dans le Tableau 5.1, ainsi que des informations complémentaires : si la couleur est discriminative pour reconnaître les objets (pour l'être humain), si les images réelles ont un point de vue à la première personne (égocentrique) ou non (3e personne).

Tableau 5.1 – Caractéristiques des jeux de données RGB-D utilisés : nombre de classes, nombre d'images d'entraînement et de test, si la couleur est suffisante pour identifier un objet, point de vue de la caméra (3e personne ou 1e personne/égocentrique).

Jeux de données	Classes	Entraînement	Test	Couleur discriminante ?	Point de vue
LINEMOD	15	50k	3000	✓	3e
T-LESS	30	50k	1000	✗	3e
RUSPA	5	50k	1388	✗	1e

5.3.1.1 LINEMOD

Le jeu de données LINEMOD comporte 15 objets tous de couleurs et formes différentes ; cela permet de les distinguer à la fois dans une image couleur et dans une image de profondeur. Nous considérons ce jeu de données comme ayant une difficulté de niveau « facile ». Il est accessible publiquement [46]. Les modèles 3D sont disponibles en ligne (et représentés sur la Figure 5.3), tout comme plusieurs ensembles d'images RGB-D annotées, réelles et synthétiques (Figure 5.4). Les images synthétiques ont été créées avec BlenderProc, et fournies à la communauté dans le cadre du challenge BOP (Hodan *et al.*, 2018), dont le but est d'améliorer les algorithmes d'estimation de pose d'objets dans des images. Les annotations contiennent donc la pose en 6D des objets, mais également leur boîte englobante 2D, qui est l'information que nous utilisons. Les images de test sont acquises par une caméra fixe, et les objets sont posés sur une table, avec quelques occultations des objets les uns par rapport aux autres. Dans ces images réelles, seulement un objet par image est annoté, ce qui demande un traitement particulier des résultats fournis par notre algorithme de détection : pour chaque image, nous ne gardons que les détections dont la classe correspond à la classe annotée.

46. https://bop.felk.cvut.cz/datasets/

FIGURE 5.3 – Les 15 objets du jeu de données LINEMOD et leurs classes associées. Image tirée de Mercier *et al.* (2019).

(a) Image RGB-D synthétique (b) Image RGB-D réelle

FIGURE 5.4 – Exemples d'images RGB-D du jeu de données LINEMOD.

5.3.1.2 T-LESS

Les 30 objets du jeu de données T-LESS correspondent à des prises et boîtiers que l'on pourrait retrouver dans un contexte industriel (Hodan *et al.*, 2017). Leur couleur est neutre et presque identique pour tous, et leurs formes sont très similaires, certains objets étant même des compositions d'autres objets (Figure 5.5). Ils sont donc difficilement reconnaissables à la fois dans les images couleur et dans les cartes de profondeur. Dans les images de test, les objets sont posés sur une table et capturés par une caméra fixe, avec seulement quelques objets visibles, qui se cachent parfois les uns les autres (Figure 5.6). Ce jeu de données a donc une difficulté « moyenne ».

De même que pour LINEMOD, des images synthétiques RGB-D générées par BlenderProc sont fournies publiquement pour le *challenge* BOP[46]. Contrairement à LINEMOD, tous les objets sont annotés dans les images réelles.

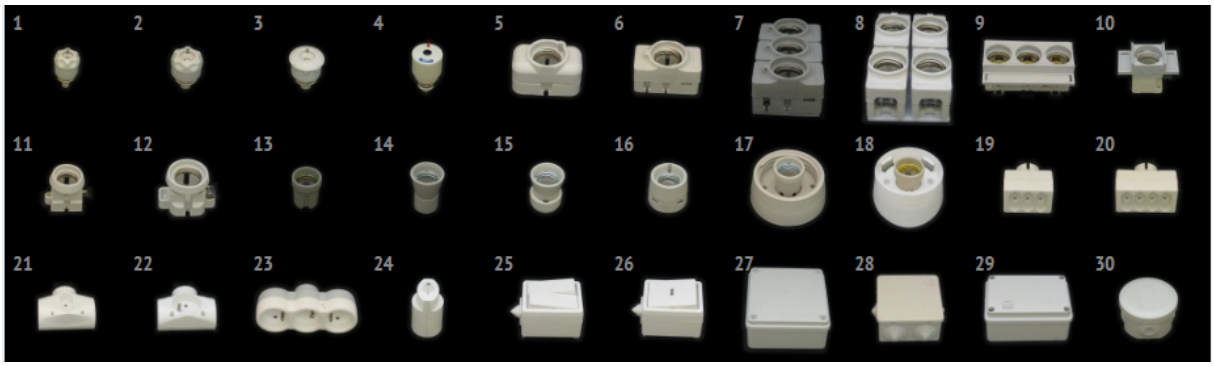
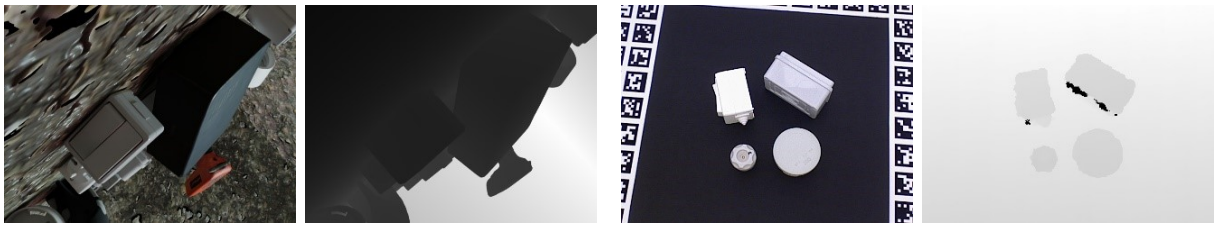


FIGURE 5.5 – Les 30 objets du jeu de données T-LESS et leurs classes associées.

5.3.1.3 RUSPA RGB-D

Il s'agit du siège de bus RUSPA introduit et utilisé dans les chapitres précédents. Pour rappel, les modèles 3D n'ont pas de texture puisqu'elle est inconnue jusqu'à l'inférence



(a) Image RGB-D synthétique (b) Image RGB-D réelle

FIGURE 5.6 – Exemples d'images RGB-D du jeu de données T-LESS.

sur les images réelles. Les images de test sont égocentriques, ce qui les rend floues avec les objets parfois sortant du champ de la caméra. Ce jeu de données est celui de difficulté « élevée ».

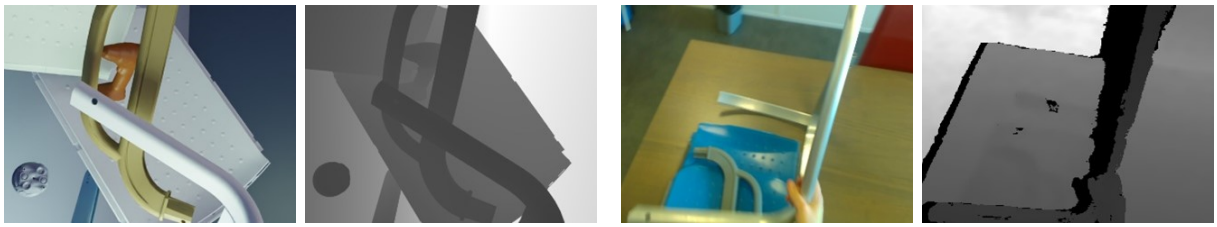
Les images d'entraînement RGB-D sont générées avec BlenderProc selon la méthode décrite dans la section 3.3.3 (Figure 5.7a). En ce qui concerne les images réelles, les images acquises précédemment dans le cadre de la détection d'objets RGB n'ont pas d'image de profondeur associée. Pour cette raison, nous avons acquis et annoté un second jeu d'images réelles, cette fois des images égocentriques et RGB-D (Figure 5.7b). Nous avons placé une caméra Intel RealSense D435i sur un prototype de casque de réalité augmentée de l'entreprise DEMS, et nous l'avons connectée à un ordinateur portable pour enregistrer les données. Nous avons acquis 8 vidéos courtes avec cette installation, parmi lesquelles 3 montrent les pièces du siège de bus en cours d'assemblage ou de désassemblage, et les 5 autres montrent chacune un seul objet, manipulé dans toutes les positions devant la caméra et partiellement hors du champ de la caméra. Nous avons ensuite enlevé les images de ces vidéos pour lesquelles aucun objet n'est visible, et nous avons annoté manuellement ces images avec des boîtes englobantes 2D. La caméra D435i comporte une caméra RGB ainsi qu'un capteur stéréoscopique infra-rouge, qui reconstruit une carte de profondeur. Cette carte de profondeur est ensuite alignée à l'image RGB automatiquement d'après les positions relatives des différents capteurs de la caméra. Un léger désalignement est encore visible, dû à la différence temporelle d'acquisition des images RGB et de profondeur, ainsi qu'au mouvement entre ces deux moments d'acquisition. Les annotations ayant été réalisées sur les images RGB, elles sont légèrement décalées par rapport aux images de profondeur. Etant donné que nous considérons une prédiction comme correcte si elle se superpose avec la vérité terrain de plus de 50 % , nous considérons que ce décalage de quelques pixels n'influe pas sur les résultats.

Ces différents jeux de données nous permettent d'évaluer la capacité d'un détecteur d'objets à généraliser à des images réelles lorsqu'il est entraîné uniquement sur des images synthétiques. Nous utilisons pour chacun 50 000 images d'entraînement (dont 1000 sont utilisées pour la validation). Nous appliquons également de l'augmentation de données, à la fois sur les images RGB et de profondeur, afin que le réseau soit confronté à des images différentes à chaque *epoch*.

5.3.2 L'augmentation de données

5.3.2.1 Augmentation forte pour les images RGB synthétiques

Afin de permettre au réseau de mieux généraliser aux images réelles, nous avons appliqué une stratégie d'augmentation de données spécifique aux images synthétiques. Pour les



(a) Image RGB-D synthétique (b) Image RGB-D réelle

FIGURE 5.7 – Exemples d'images RGB-D du jeu de données RUSPA.

images RGB, nous avons appliqué ici l'augmentation « forte » présentée dans le chapitre précédent, section 4.3.2.

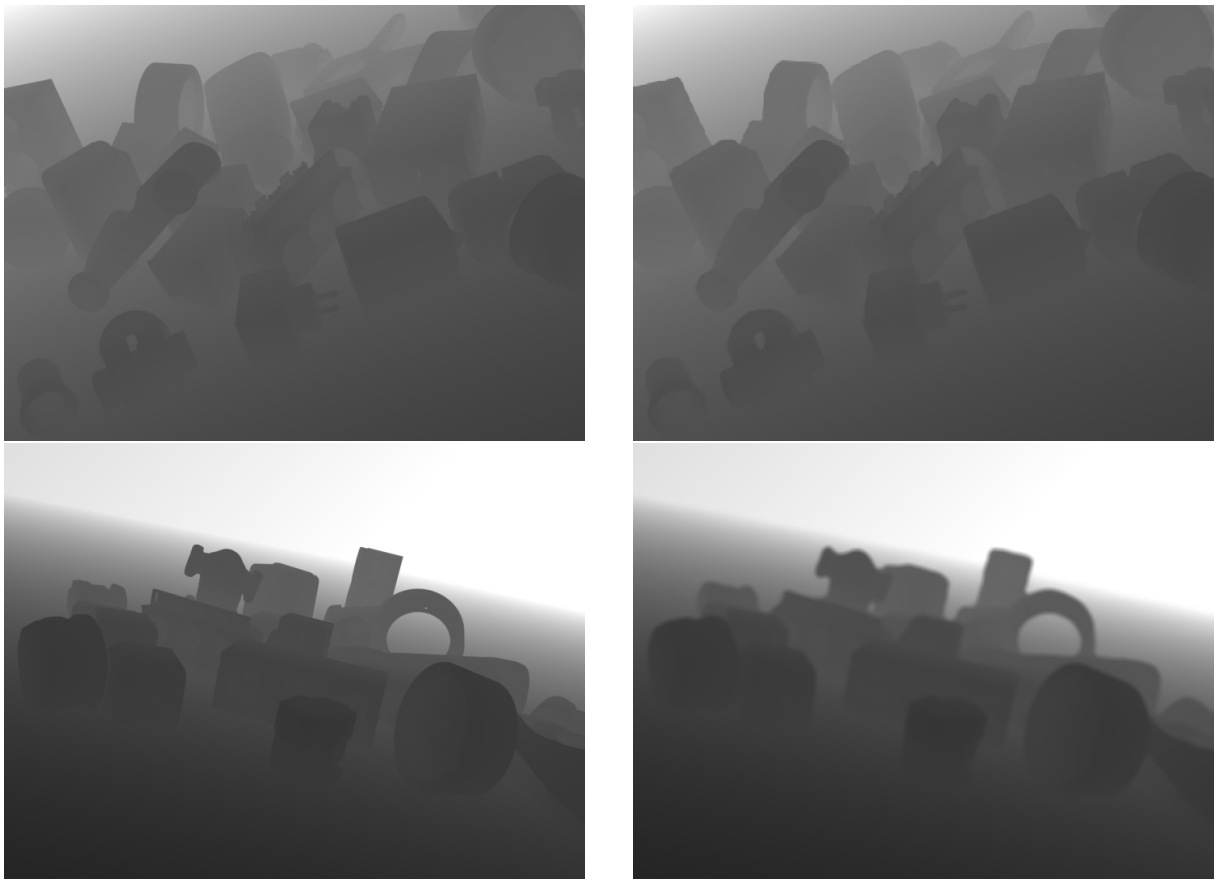
5.3.2.2 Augmentation forte pour les images de profondeur synthétiques

Dans beaucoup d'applications utilisant des images de profondeur, il est souhaitable de ne pas altérer la valeur des pixels, afin de connaître la distance exacte aux objets. Dans notre cas, nous n'avons pas besoin de cette information de distance, c'est pourquoi nous appliquons à la fois des opérations géométriques et colorimétriques sur les images de profondeur, comme pour les images couleurs. De plus, les images de profondeur synthétiques sont particulièrement nettes, avec les arrêtes des objets bien définies, ce qui les rend très différentes des images réelles et justifie ici encore une augmentation particulièrement « forte » de ces images.

Nous avons défini expérimentalement une série d'augmentations permettant d'altérer l'apparence des images de profondeur synthétiques lors de l'entraînement, afin qu'elles soient plus réalistes. Les mêmes opérations géométriques que pour les images RGB sont appliquées, de façon synchronisée afin que la couleur et la profondeur correspondent toujours après augmentation. Pour les opérations colorimétriques, nous appliquons les opérations suivantes : une diminution de la définition suivie de flou médian (afin de produire un aspect moins net), du flou médian avec un noyau plus large, du flou au choix parmi du flou moyen, Gaussien ou de mouvement. Ces opérations et leurs paramètres sont détaillés dans le Tableau 5.2.

TABLEAU 5.2 – Séquence d'opérations colorimétriques pour une augmentation« forte » des cartes de profondeur synthétiques, avec des fonctions de la bibliothèque logicielle *albumentations* (Buslaev *et al.*, 2020).

Opérations et valeurs de paramètres	**Probabilité**
Downscale(scale_min=0,25, scale_max=0,5) **Suivi de** MedianBlur(blur_limit=[3, 5])	0,6
MedianBlur(blur_limit=[9, 15])	0,1
Choisir une opération parmi GaussianBlur(blur_limit=[5, 9], sigma_limit=0) Blur(blur_limit=[3, 7]) MotionBlur(blur_limit=[3, 5])	0,7

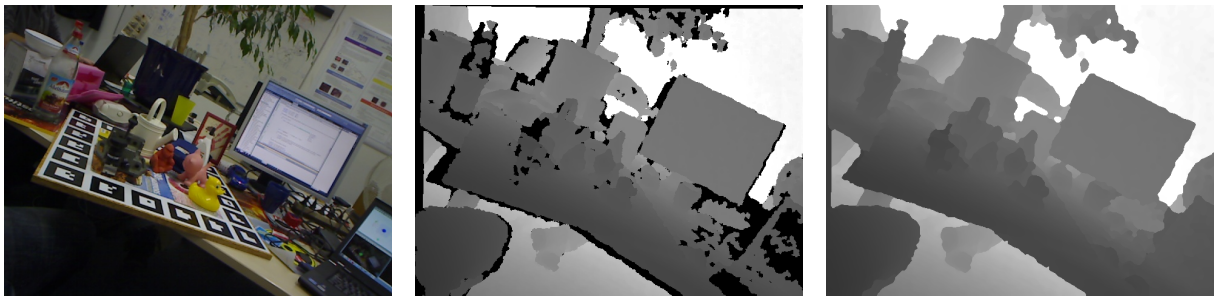


(a) Avant augmentation. (b) Après augmentation

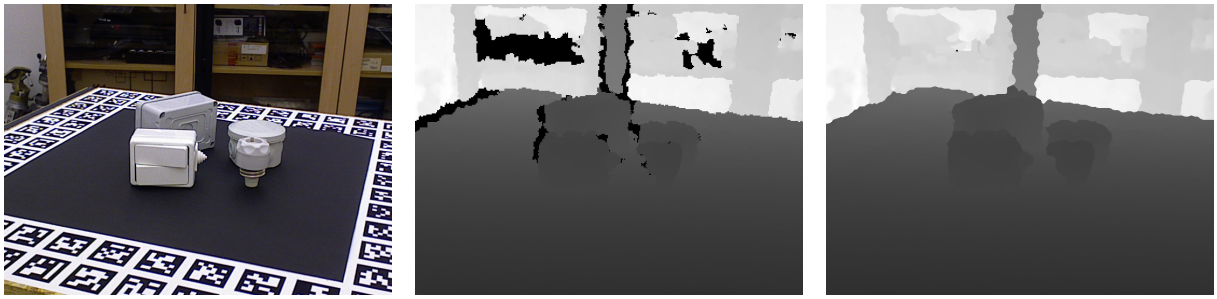
FIGURE 5.8 – Exemples d'images de profondeur synthétiques du jeu de données T-LESS, avec et sans augmentation de données.

5.3.2.3 Complétion des images de profondeur

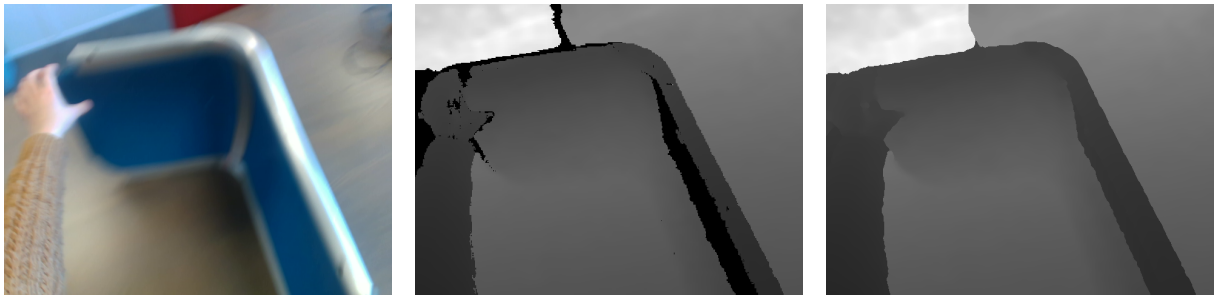
Malgré ces augmentations, les images synthétiques sont toujours très différentes des images réelles, notamment en raison des défauts comme les pixels manquants que nous ne simulons pas. Après avoir étudié différentes approches proposées dans la littérature (voir paragraphe 2.4.4), nous avons choisi d'utiliser la méthode par traitement d'image classique afin de compléter les images de profondeur réelles à l'inférence (Ku *et al.*, 2018). Bien que cette méthode soit destinée à l'amélioration de la qualité d'images de type LiDAR pour l'analyse de scènes urbaines, nous avons trouvé qu'elle peut s'appliquer également aux images de capteurs Primesense ou Intel RealSense. De plus, cette méthode est extrêmement rapide, de l'ordre de la dizaine de millisecondes par image pour un CPU d'ordinateur standard. Nous l'avons donc adoptée pour améliorer la qualité de images de profondeur réelles lors de l'inférence, avant de les donner au détecteur d'objets. Au contraire, la méthode de colorisation par optimisation (Levin *et al.*, 2004) est trop lente pour être utilisée dans notre contexte, tout comme les méthodes par apprentissage profond, qui demandent en plus des images représentatives du domaine cible, ce dont nous ne disposons pas. Le résultat pour les bases de données LINEMOD, T-LESS et RUSPA est montré sur les Figures 5.9a, 5.9b et 5.9c respectivement. Les résultats étant satisfaisants, nous ne nous sommes pas penchés sur l'adaptation de cet algorithme à nos données et avons utilisé l'implémentation disponible publiquement, sans la modifier.



(a) LINEMOD - RGB, profondeur, profondeur complétée.



(b) T-LESS - RGB, profondeur, profondeur complétée.



(c) RUSPA - RGB, profondeur, profondeur complétée.

FIGURE 5.9 – Exemples des bases de données étudiées, avec la carte de profondeur complétée par traitement d'images classique (Ku *et al.*, 2018).

L'algorithme complet est résumé ci-dessous (Algorithme 1). Les opérations et noyaux associés ont été choisis par expérimentation avec le *benchmark* de complétion de la profondeur KITTI. Initialement, une carte de profondeur contient des valeurs proches de 0 pour les objets les plus proches, et des valeurs plus grandes pour les objets les plus lointains. Cette image est d'abord inversée tout en gardant les pixels manquants à 0, afin de tirer parti de l'implémentation des opérations morphologiques dans OpenCV : les valeurs les plus grandes prennent le dessus sur les valeurs les plus petites. En travaillant avec les distances inversées, les pixels manquants vont être complétés de façon à correspondre aux objets voisins au premier plan, et non à l'arrière-plan. Lors de cette inversion, il est nécessaire de garder un écart avec la valeur la plus grande, afin que les pixels les plus lointains ne soient pas mis à 0 une fois inversés, ce qui les confondrait avec les pixels manquants. Dans la méthode originale, les cartes de profondeur à compléter prennent des valeurs de distance de 0m à 80m et l'écart pris est de 20m. Dans notre cas, les distances couvrent d'environ 10cm à 10m, nous prenons donc un écart de 5cm afin d'assurer la bonne séparation des pixels manquants (à 0) et des pixels les plus proches (à 10cm). Une opération de dilatation est ensuite appliquée afin de compléter les pixels manquants proches des pixels valides. Afin de prendre en compte la nature du capteur LiDAR, les auteurs défi-

nissent et comparent 4 noyaux de formes différentes, et choisissent finalement le noyau en forme de diamant 5×5, qui obtient les meilleurs résultats. Les petits trous restant dans l'image sont complétés avec une opération de fermeture de noyau complet 5×5, ce qui permet de connecter les bords des objets proches les uns des autres. La fermeture de ces contours créant de nombreuses petites zones de pixels manquants, celles-ci sont remplies par une dilatation avec un noyau complet de taille 7×7, après avoir calculé le nouveau masque des pixels manquants afin de ne pas altérer les pixels connus. Ensuite, les auteurs prolongent les objets hauts (tels que des arbres ou des poteaux) jusqu'en haut de l'image couleur, dont la définition n'est pas la même que celle de l'image LiDAR. Pour cela, ils extrapolent les valeurs de la ligne supérieure de pixels jusqu'au haut de l'image. Dans notre cas, les images RGB et de profondeur ont la même définition, cette étape n'a donc pas d'effet. La dernière étape consiste à remplir les pixels manquants correspondant à des trous plus larges, qui n'auraient pas été complétés jusque-là. Ces valeurs manquantes sont extrapolées à partir des pixels connus les plus proches, avec une dilatation de noyau complet de 31×31 pixels. A nouveau, seuls les pixels manquants sont remplis, les pixels connus n'étant pas modifiés. Enfin, les valeurs aberrantes créées par les opérations de dilatation successives sont supprimées en appliquant du flou médian avec un noyau de taille 5×5, ce qui préserve les bords, puis un flou Gaussien également de noyau 5×5 est appliqué. Ce dernier permet de lisser les surfaces localement, mais a également pour effet d'arrondir les bords des objets. Pour éviter cela, un filtre bilatéral peut être utilisé pour remplacer le flou Gaussien : au lieu de flouter les pixels en se basant uniquement sur leur proximité spatiale, le filtre bilatéral considère également la différence entre leurs intensités, afin de ne flouter que les pixels d'intensités similaires. Bien que légèrement plus lent, le filtre bilatéral préserve bien mieux les bords des objets. L'image de profondeur obtenue est finalement inversée à nouveau pour ramener les objets les plus proches aux plus petites valeurs de pixels.

Algorithme 1 Complétion des pixels manquants dans les images de profondeur.

Entrées : une carte de profondeur brute comportant des pixels manquants repérés par un masque.

Etape 1 : inversion des pixels connus en gardant une distance minimum correspondant à 5cm.

Etape 2 : dilatation avec un kernel diamant 5×5.

Etape 3 : fermeture avec un kernel complet 5×5.

Etape 4 : calcul d'un nouveau masque de pixels manquants, puis dilatation avec un kernel complet 7×7 et mise à jour des pixels manquants uniquement.

Etape 5 : extension des valeurs de la ligne supérieure jusqu'au haut de l'image pour atteindre la définition de l'image couleur.

Etape 6 : dilatation avec un kernel complet 31×31 et mise à jour des pixels manquants uniquement.

Etape 7 : flou médian avec un kernel 5×5, puis flou Gaussien avec un kernel 5×5 ou filtre bilatéral.

Etape 8 : inversion de l'image.

Sorties : l'image de profondeur dont les pixels manquants ont été complétés.

5.4 Expériences et résultats

Dans cette partie, nous détaillons les expériences et analyses menées concernant la détection d'objets dans des images RGB-D. Nous nous intéressons en particulier aux questions suivantes :

- La combinaison des modalités améliore-t-elle la performance par rapport à l'une ou l'autre des modalités seules ?
- Les modalités contribuent-elles de façon équitable ?
- Comment la base de données influence-t-elle l'apprentissage et les résultats ?
- La problématique d'écart à la réalité est-elle la même avec des images RGB-D qu'avec des images RGB ?

5.4.1 Procédure d'évaluation

Dans cette partie, nous avons entraîné le réseau MobileNetV3 SSD *Large* (dénoté SSD pour la suite) avec des images RGB et des images de profondeur, afin d'établir une performance de référence avant d'évaluer notre architecture fusionnant les deux modalités. Les réseaux résultants sont nommés RGB SSD, Prof. SSD, et RGB-D SSD. Nous avons comparé la performance de SSD uni-modal avec celle de l'architecture RetinaNet (Lin *et al.*, 2020), un réseau de neurones récent pour la détection d'objets inspiré de SSD. Nous avons entraîné RetinaNet sur les images RGB et de profondeur séparément.

La performance est mesurée comme précédemment avec la précision moyenne (mAP) calculée avec tous les points de la courbe précision-rappel [47]. Les réseaux sont entraînés sur les images synthétiques uniquement pour chacun des jeux de données décrits précédemment, et les images réelles ne sont utilisées que comme ensemble de test. Pour chaque expérience, le réseau peut être initialisé de façon aléatoire, ou à partir d'un pré-entraînement sur ImageNet. Nous donnons également la performance de validation, mesurée sur 1000 images synthétiques pour chaque base de données, afin de donner un aperçu de l'écart de domaine : une performance bien inférieure sur les images de test que sur les images de validation indique un manque de généralisation, probablement dû à la différence de domaine dans notre cas. Pour les images de profondeur, nous calculons la performance à la fois sur les images brutes et sur les images complétées par la méthode de traitement d'images classique, détaillée dans le paragraphe 5.3.2.3.

5.4.2 Détection d'objets uni-modale

5.4.2.1 Détails d'entraînement

A la suite des résultats précédents, nous avons choisi le modèle MobileNetV3 SSD *Large* comme architecture de référence pour la détection d'objets. Une implémentation officielle PyTorch ayant été publiée dans la bibliothèque *torchvision* après nos travaux du chapitre précédent, nous l'avons utilisée pour la suite. De même, nous avons utilisé l'implémentation *torchvision* de RetinaNet. Au-delà de rééquilibrer les exemples positifs et négatifs dans sa fonction de coût, une différence majeure est l'extracteur de caractéristiques utilisé par RetinaNet : ResNet50-FPN. Ce modèle extrait des caractéristiques à

47. Il s'agit de la même métrique que pour l'évaluation de SSD avec les images RGB du jeu de données T-LESS. L'implémentation du réseau n'est cependant pas la même, ce qui explique des résultats différents.

différentes échelles grâce à un *Feature Pyramid Network* (FPN), ce qui le rend beaucoup plus gros que SSD. De plus, il s'appuie sur ResNet50, un extracteur de caractéristiques composé de 50 blocs, ce qui n'est pas la plus petite version de ResNet. Il est donc attendu de RetinaNet une meilleure performance que SSD, pour un temps de calcul bien supérieur, bien que RetinaNet soit parfois considéré comme un « petit » réseau dans l'univers de l'apprentissage profond.

Tous les réseaux ont été entraînés avec une taille de *batch* de 64 images, un taux d'apprentissage de $1e^{-3}$, un *momentum* de 0,9, et un *weight decay* de $5e^{-4}$. Les réseaux sont entraînés à la fois depuis une initialisation aléatoire et depuis un pré-entraînement sur le jeu de données ImageNet. Dans le cas de SSD, l'architecture du réseau pré-entraîné ou non varie légèrement. En effet, les auteurs de (Howard *et al.*, 2019) ont observé que pour un jeu de données avec bien moins de classes que les 1000 classes d'ImageNet, il était bénéfique de réduire la taille des dernières couches du réseau.

TABLEAU 5.3 – Résultats de détection d'objets uni-modale sur LINEMOD, T-LESS et RUSPA, avec initialisation aléatoire ou pré-entraînement avec ImageNet. La performance de test pour la profondeur est donnée sur les images brutes/complétées.

mAP (%)	Pré-entraînement ImageNet	LINEMOD		T-LESS		RUSPA	
		Valid.	Test	Valid.	Test	Valid.	Test
RGB RetinaNet	✓	81,2	**89,2**	54,3	51,1	80,1	**41,7**
	✗	56,6	55,7	14,8	11,3	33,7	23,4
Prof. RetinaNet	✓	81,7	51,3/**82,6**	46,7	22,2/25,8	80,8	**10,7**/9,1
	✗	44,6	0,2/12,4	6,9	4,4/6,3	50,6	6,1/6,8
RGB SSD	✓	66,4	68,9	46,7	42,5	74,1	24,5
	✗	66,8	78,4	54,4	**53,4**	54,4	29,0
Prof. SSD	✓	67,5	47,1/58,8	44,1	16,0/19,9	79,2	3,2/2,7
	✗	71,6	47,5/78,0	47,5	23,5/**30,4**	68,4	6,4/5,8

5.4.2.2 Résultats

Le Tableau 5.3 présente les résultats des réseaux SSD et RetinaNet avec une seule modalité en entrée. La performance est donnée pour les images de validation (synthétiques), les images de test (réelles) brutes, et les images de test dont la profondeur a été complétée (uniquement pour les réseaux entraînés avec des images de profondeur).

De façon générale, on observe que les réseaux entraînés avec des images RGB détectent mieux les objets que les réseaux entraînés avec des images de profondeur, pour tous les jeux de données et architectures de réseaux. Pour les images RGB de LINEMOD et T-LESS, la performance de test est à peu près équivalente à celle de validation ; cela suggère qu'il n'y a pas de problème d'écart de domaine entre les deux. En revanche, pour les images de profondeur, il y a systématiquement une chute de la performance, le réseau n'arrivant pas bien à généraliser aux images réelles de profondeur brutes. Lorsque ces images de profondeur sont complétées, elles sont beaucoup plus ressemblantes aux images synthétiques, ce qui réduit suffisamment l'écart de domaine pour améliorer la performance.

Pour LINEMOD, les objets ont des formes très différentes, ce qui permet d'atteindre plus de 80 % de mAP avec RetinaNet (pour un réseau pré-entraîné avec ImageNet), et 78 % avec SSD (lorsque le réseau n'est pas pré-entraîné avec ImageNet).

Pour T-LESS, les réseaux entraînés sur les images RGB atteignent à peine plus de 50 %, et à peine plus de 20 % pour les réseaux entraînés avec des images de profondeur

et testés sur la profondeur brute. Ces résultats s'améliorent légèrement en complétant les images de profondeur, jusqu'à 30 % pour SSD, et 25 % pour RetinaNet. Cependant, la performance maximale semble être très limitée, comme en atteste la performance en validation qui dépasse à peine 50 % pour SSD comme pour RetinaNet pour les images RGB, et un peu moins pour les images de profondeur. On voit ici le manque de diversité dans la géométrie et dans la couleur des objets de T-LESS, qui rend plus complexe leur identification, même sur des images synthétiques très similaires aux images d'entraînement. Enfin, il est ici à noter que RGB SSD permet d'obtenir une précision légèrement supérieure à RetinaNet, 53,4 % contre 51,1 % ; cela laisse supposer que RetinaNet a une architecture trop complexe pour le peu de texture et de variabilité dans les objets de la base de données T-LESS.

En ce qui concerne RUSPA, un écart important entre les performances sur images synthétiques et réelles est déjà présent pour les images RGB, indiquant une difficulté à généraliser aux images réelles. De plus, le réseau ne généralise pas du tout aux images de profondeur, donnant une précision au maximum de 10 % à la fois avec SSD et RetinaNet. Ces résultats montrent bien la complexité de la tâche sur ces données.

5.4.3 Détection d'objets multimodale

Nous avons comparé les 4 méthodes de fusion proposées précédemment pour l'architecture MobileNetV3 SSD : concaténation, moyenne, addition, et convolution. La fusion par moyenne n'apparait pas dans les résultats suivants car les réseaux n'ont pas convergé durant l'entraînement : en moyennant les deux modalités, le réseau ne parvient pas à extraire d'information utile pour la détection d'objets. Les réseaux entraînés sont notés RGB-D SSD (Concat) pour la concaténation, RGB-D SSD (Add) pour l'addition, et RGB-D SSD (Conv) pour la convolution.

5.4.3.1 Détails d'entraînement

L'implémentation de l'architecture proposée est une extension du réseau précédent MobileNetV3 SSD. Nous avons ajouté un second extracteur de caractéristiques MobileNetV3 en parallèle du premier, identique mais dont les poids ne sont pas partagés, et deux couches de fusion associées aux couches C4 et C5, respectivement. L'architecture SSD reste inchangée, sauf dans le cas de la fusion par concaténation, qui crée des couches de caractéristiques plus larges que dans le cas uni-modal. Les premières couches de SSD suivant la fusion ont donc une profondeur deux fois plus grande, puis le reste du réseau reste inchangé.

Tous les réseaux ont été entraînés avec une taille de *batch* de 32 images, un taux d'apprentissage de $1e^{-3}$, un *momentum* de 0,9, et une régularisation par *weight decay* de coefficient $5e^{-4}$. Ces hyperparamètres ont été choisis d'après la performance sur l'ensemble de validation. Les réseaux sont initialisés de façon aléatoire ou à partir d'un pré-entraînement sur ImageNet.

Les résultats présentés ici sur les images de test correspondent aux images RGB-D dont l'image de profondeur est complétée, puisque nous avons observé précédemment que cela réduit fortement l'écart de domaine entre les images synthétiques et réelles.

5.4.3.2 Résultats

Les résultats sont rassemblés dans le Tableau 5.4. Pour LINEMOD et T-LESS, toutes les méthodes de fusion améliorent la performance par rapport au réseau à une seule modalité. De plus, les réseaux généralisent bien mieux lorsque l'extracteur de caractéristiques n'est pas pré-entraîné avec le jeu de données ImageNet.

Pour LINEMOD, la précision obtenue augmente de 78,4 % (RGB SSD) à 81,8 % avec une fusion par concaténation, 83,5 % avec une convolution, et 84,5 % avec une addition. S'il n'atteint pas encore les 89,2 % de mAP de RetinaNet, le réseau SSD en est très proche pour une fraction du temps et coût de calcul, comme nous le verrons par la suite. Avec T-LESS, la performance augmente de 53,4 % avec RGB SSD, à 54,5 % avec la fusion par convolution, 56,7 % avec une concaténation, et enfin 56,8 % avec une fusion par addition. Malgré le peu de détails contenus dans les images de profondeur, elles apportent tout de même un complément d'information utile à l'image RGB.

Pour RUSPA en revanche, la difficulté d'apprentissage sur les images de profondeur semble tirer vers le bas la performance de tous les réseaux RGB-D SSD. Malgré une bonne performance sur les images de validation (synthétiques), le réseau ne généralise absolument pas aux images réelles de l'ensemble de test, comme c'était déjà le cas avec Prof. SSD, l'architecture entraînée uniquement avec les images de profondeur.

TABLEAU 5.4 – Résultats de détection d'objets multimodale pour LINEMOD, T-LESS et RUSPA, avec initialisation aléatoire ou pré-entraînement avec ImageNet. Les images de profondeur de test sont complétées.

mAP (%)	Pré-entraînement ImageNet	LINEMOD		T-LESS		RUSPA	
		Valid.	Test	Valid.	Test	Valid.	Test
RGB-D SSD (Concat)	✓	74,2	74,0	56,0	36,2	79,3	6,4
	✗	75,7	81,8	58,6	56,7	75,7	5,4
RGB-D SSD (Add)	✓	74,8	70,7	56,0	35,1	81,4	5,5
	✗	75,7	**84,3**	56,6	**56,8**	71,7	4,7
RGB-D SSD (Conv)	✓	76,9	71,1	59,2	42,3	81,6	**7,4**
	✗	74,9	83,5	58,3	54,5	68,4	3,3

5.4.3.3 Analyse des images de profondeur de la base de données RUSPA

Bien que le capteur Intel RealSense, qui a servi à capturer les images RGB-D réelles de RUSPA, soit similaire à un capteur Primesense, qui a produit les images réelles de LINEMOD et T-LESS, les images de profondeur résultantes présentent des différences. Quelques défauts visibles dans les images réelles de RUSPA sont illustrés sur la Figure 5.10.

Tout d'abord, la scène et le point de vue ne sont pas les mêmes : les objets de T-LESS et LINEMOD ressortent bien par rapport au fond, alors que les images RUSPA présentent parfois très peu de contraste entre les objets d'intérêt et la table sur laquelle ils sont posés. De plus, la complétion des images de profondeur par la méthode de traitement d'image classique n'est pas aussi efficace dans les images RUSPA, et certaines régions sont mal complétées. En particulier, toute la région manquante tend à être complétée avec la valeur de profondeur la plus petite du voisinage (donc l'objet le plus proche), ce qui élargit les contours des objets et tend à fondre les objets dans leur voisinage. En modifiant la forme des objets dans les images réelles, la représentation apprise par le réseau avec les images synthétiques ne correspond plus, ce qui l'empêche de généraliser au domaine réel.

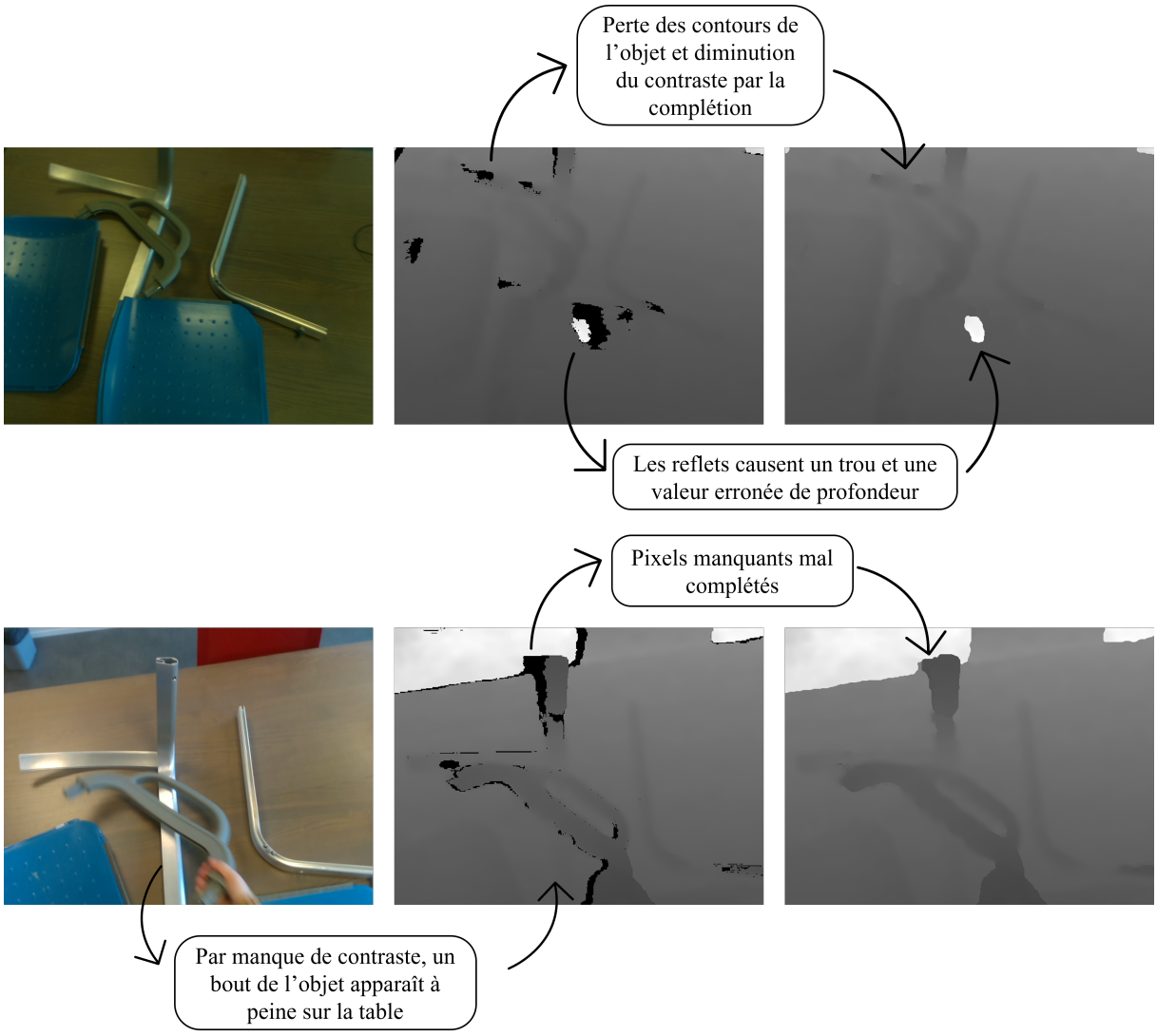


FIGURE 5.10 – Illustration des défauts dans les images de profondeur de RUSPA et leur version complétée.

5.4.4 Analyse de l'architecture des réseaux

Comme observé précédemment, le réseau RetinaNet permet d'obtenir en général une meilleure performance que SSD sur les images de validation, signe d'une capacité d'apprentissage supérieure. Cette performance est également supérieure à celle de SSD sur les images réelles pour la plupart des modalités et jeux de données. Cependant, RetinaNet est également un réseau de taille bien supérieure à SSD, comme l'indique le Tableau 5.5. Le nombre de paramètres de RetinaNet est supérieur à celui de SSD d'un ordre de grandeur, y compris pour l'architecture multimodale. Cela se traduit dans le nombre d'opérations *Mult-Adds*, de l'ordre du millier pour le plus gros modèle SSD proposé, contre 20 fois plus pour RetinaNet. Enfin, le temps d'inférence de SSD est environ divisé par 3 par rapport à celui de RetinaNet.

TABLEAU 5.5 – Analyse de la performance des modèles : nombre de paramètres, nombre d'opérations *Mult-Adds* (en millions d'opérations), et durée de l'inférence en ms pour une image (moyenne de 10 répétitions).

	Paramètres	Mult-Adds (M)	Durée de l'inférence (ms)
LINEMOD			
RGB SSD	$2{,}4 \cdot 10^6$	433	30,9
RGB-D SSD (Concat)	$4{,}31 \cdot 10^6$	858	36,9
RGB-D SSD (Add)	$4{,}02 \cdot 10^6$	802	34
RGB-D SSD (Conv)	$5{,}41 \cdot 10^6$	1220	36,2
RGB Retinanet	$3{,}25 \cdot 10^7$	20 930	92
T-LESS			
RGB SSD	$2{,}61 \cdot 10^6$	463	43,8
RGB-D SSD (Concat)	$4{,}62 \cdot 10^6$	917	42,6
RGB-D SSD (Add)	$4{,}23 \cdot 10^6$	832	43,4
RGB-D SSD (Conv)	$5{,}62 \cdot 10^6$	1250	43,5
RGB Retinanet	$3{,}28 \cdot 10^7$	21 590	94,6
RUSPA			
RGB SSD	$2{,}26 \cdot 10^6$	413	27,1
RGB-D SSD (Concat)	$4{,}1 \cdot 10^6$	819	37,7
RGB-D SSD (Add)	$3{,}88 \cdot 10^6$	782	39,7
RGB-D SSD (Conv)	$5{,}27 \cdot 10^6$	1200	41,8
RGB Retinanet	$3{,}23 \cdot 10^7$	20 480	91,3

5.4.5 Analyse de robustesse

Les réseaux de neurones à plusieurs branches, qui se nourrissent de plusieurs types d'information, présentent souvent un défaut commun : lorsque l'une des entrées est vide, la performance chute drastiquement. En effet, alors que le réseau se base sur une combinaison de caractéristiques, il n'est plus capable de prédire correctement la sortie lorsqu'il lui manque la moitié des informations. Dans une application réelle, cette situation est susceptible de se produire, par exemple si l'un des capteurs de la caméra tombe en panne ou si la reconstruction de la profondeur prend plus de temps. Dans ce cas, le comportement attendu serait que le réseau effectue une prédiction seulement à partir de la modalité disponible, ce qui se traduirait par une perte de performance limitée par rapport à la situation « idéale » dans laquelle les deux modalités sont disponibles. Dans cette partie, nous réalisons une analyse de notre réseau RGB-D SSD (Conv) dans le cas où l'une des modalités viendrait à manquer, et nous proposons une méthode d'entraînement qui n'engendre aucun coût supplémentaire, tout en permettant d'améliorer la robustesse du réseau obtenu.

5.4.5.1 Méthode

Pour obtenir un réseau multimodal robuste, nous nous sommes inspirés de l'approche ModDrop (Neverova *et al.*, 2016). Dans cette publication, les modalités sont parfois annulées pendant l'entraînement, afin que le réseau apprenne à « faire sans » lorsque cela devient nécessaire. Contrairement à l'article original qui applique cette approche au moment de la fusion des modalités, nous l'appliquons dès l'entrée du réseau en simulant une image « vide », composée uniquement de zéros. Lors de l'entraînement, nous comparons plusieurs approches : enlever la composante RGB de 10 % des images d'entraînement ;

enlever la composante de profondeur de 10 % des images d'entraînement ; ou bien enlever l'une des deux composantes de façon aléatoire, toujours pour 10 % des images. Cette analyse pourra être approfondie en faisant varier le pourcentage d'images manquantes.

5.4.5.2 Résultats

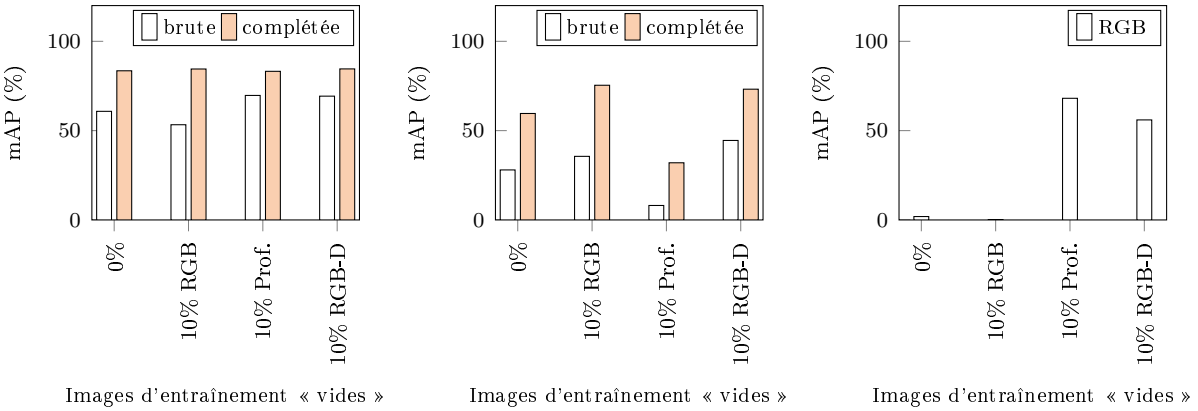
Les résultats sont montrés sur la Figure 5.11. Nous avons analysé la performance de notre réseau SSD multimodal lorsque seule l'image RGB est disponible (Figure 5.11c), et lorsque seule l'image de profondeur est disponible (Figure 5.11b), pour la base de données LINEMOD. Nous avons également représenté la performance lorsque les deux images sont disponibles, ce qui correspond au cas normal (Figure 5.11a). Nous avons comparé les résultats obtenus lorsque le modèle est entraîné de façon classique (aucune image « vide »), en enlevant 10 % des images RGB, en enlevant 10 % des images de profondeur, et en enlevant 10 % des images RGB ou de profondeur.

Tout d'abord, lorsque le modèle est entraîné avec toutes les images (0 % d'images d'entraînement « vides »), nous pouvons observer qu'il effectue ses prédictions en se basant en majorité sur les images de profondeur. En effet, la performance baisse un peu si l'on enlève toutes les images RGB à l'inférence, et elle tombe à presque 0 si l'on enlève les images de profondeur.

Lorsque le modèle est entraîné avec 10 % d'images en moins (peu importe la modalité) et testé sur toutes les images (scénario classique dans lequel les deux modalités sont disponibles), la performance ne diminue pas lorsque les images de profondeur sont complétées. Cela indique que ce mode d'entraînement n'est pas néfaste pour la performance dans les conditions classiques. Lorsque l'image de profondeur n'est pas complétée et que l'on enlève une partie des images de profondeur, cette performance est même meilleure ; cela semble indiquer que le réseau s'appuie plus fortement sur l'image RGB. Enlever une partie des images de profondeur pendant l'entraînement permet même d'obtenir une meilleure performance lorsque les images de profondeur ne sont pas complétées, ce qui montre que le réseau s'appuie moins dessus.

Ensuite, lorsque l'on enlève des images RGB pendant l'entraînement, le modèle semble s'appuyer totalement sur la profondeur, puisque sa précision diminue très peu lorsqu'il est testé sans images RGB, et chute à 0 lorsqu'il est testé sans images de profondeur. Au contraire, en enlevant 10 % des images de profondeur pendant l'entraînement, le réseau semble se baser principalement sur les images RGB ; la performance lorsque l'on teste sur les images RGB uniquement est assez proche de la performance classique avec les deux modalités. En revanche, si l'on enlève les images RGB à l'inférence, la performance ne chute pas à 0 comme c'était le cas précédemment. Le réseau continue donc à utiliser l'image de profondeur comme source d'information. Pour terminer, lorsque l'on enlève à la fois des images RGB et de profondeur pendant l'entraînement, la performance diminue très peu lorsque l'une ou l'autre des modalités est manquante. A nouveau, le réseau prédit mieux les objets lorsque les images de profondeur sont présentes, signe qu'il s'appuie encore plus sur la profondeur que la couleur.

En conclusion, entraîner le modèle en enlevant 10 % des images RGB ou de profondeur, de façon aléatoire, permet bien d'obtenir un modèle plus robuste à la perte d'une modalité, sans avoir d'effet négatif dans le cas normal multimodal.



(a) Test sur RGB-D. (b) Test sur profondeur. (c) Test sur RGB.

FIGURE 5.11 – Analyse de robustesse de RGB-D SSD (Conv) avec LINEMOD. Le modèle est entraîné avec toutes les images (0% d'images « vides »), 10% d'images RGB « vides » (10% RGB), 10% d'images de profondeur vides (10% Prof.) ou 10% d'images « vides » de l'une ou l'autre des modalités (10% RGB-D). La performance est évaluée sur les images de test (a) RGB-D, (b) uniquement les images de profondeur, (c) uniquement les images couleur. Les images de profondeur de test sont brutes ou complétées.

5.5 Conclusion

Dans ce chapitre, nous avons présenté nos différents travaux sur la fusion de modalités comme stratégie pour améliorer la détection d'objets, dans le cas particulier des images RGB-D. Nous avons présenté l'architecture multimodale proposée, qui combine deux extracteurs de caractéristiques identiques mais de poids non partagés, et adaptés à l'inférence en temps réel avec peu de capacités de calcul. Un unique réseau de détection d'objets SSD est appliqué à la suite des couches de caractéristiques fusionnées provenant des deux modalités, limitant ainsi la taille totale du modèle.

Nous avons montré que plusieurs facteurs relatifs au jeu de données jouent sur la performance : la diversité des objets, tant dans leur couleur que leur géométrie ; la qualité des images de profondeur ; le type de bruit et de défauts présents dans ces images de profondeur. Nous avons ainsi obtenu des résultats satisfaisants sur les deux jeux de données publics LINEMOD et T-LESS, dont les images sont capturées par des caméras fixes dans un environnement contrôlé, alors que les images de notre jeu de données industriel capturé en conditions plus réalistes présentent des défis supplémentaires, encore non résolus. Pour LINEMOD et T-LESS, l'écart de domaine entre les images réelles et synthétiques est comblé différemment selon la modalité. En ce qui concerne les images RGB, une augmentation de données importante appliquée aux images d'entraînement est nécessaire. Pour les images de profondeur, l'augmentation des images d'entraînement n'est pas suffisante ; nous avons donc complété les pixels manquants dans les images réelles grâce à des opérations morphologiques, une approche suffisamment rapide pour nos besoins qui améliore fortement les résultats. En revanche, ces opérations morphologiques ne sont pas adaptées aux images de la base de données RUSPA, ce qui ne permet pas au réseau de généraliser aux images de profondeur réelles.

Enfin, nous avons proposé une méthode d'entraînement spécifique pour rendre le réseau multimodal plus robuste à une modalité manquante. En remplaçant l'une ou l'autre des images d'une paire RGB-D par une image vide, pour un petit nombre d'images d'en-

traînement, le réseau apprend à compenser cette absence. Ainsi, la perte de performance lorsqu'une image est manquante au moment de l'inférence est significativement réduite par rapport à un entraînement classique, comme le montrent nos expériences avec le jeu de données LINEMOD. De plus, l'approche proposée ne demande aucune opération supplémentaire pendant l'entraînement, et ne réduit pas la performance obtenue lorsque les deux modalités sont bien présentes.

Alors que nous avons observé une meilleure performance lorsque l'extracteur de caractéristiques n'est pas pré-entraîné sur le jeu de données ImageNet, il est tout de même naturel de se demander si un pré-entraînement différent ne pourrait pas fournir une meilleure base pour nos détecteurs d'objets. Plusieurs approches s'offrent à nous pour cela : entraîner les extracteurs de caractéristiques séparément pour chacune des modalités, ou bien conjointement avec des images RGB-D d'une ou plusieurs bases de données publiques ; apprendre ces caractéristiques initiales avec une tâche de classification ou à l'aide d'une tâche prétexte ; utiliser des images synthétiques et/ou des images réelles d'un autre jeu de données RGB-D. Idéalement, ce pré-entraînement donnerait à l'extracteur de caractéristiques une grande capacité de généralisation à d'autres images telles que celles de RUSPA.

Conclusion générale et perspectives

Dans cette thèse, nous avons abordé la détection d'objets industriels dans des images égocentriques en utilisant des algorithmes d'apprentissage profond. Nous avons développé différentes briques logicielles qui, une fois assemblées, permettent de générer un jeu de données synthétiques et d'entraîner un réseau de neurones à convolutions. Nous avons également étudié plusieurs réseaux de neurones pour cette tâche, et nous avons proposé une architecture afin d'utiliser l'information présente dans une carte de profondeur. Alors que nous nous sommes concentrés sur l'utilisation des données à disposition sous forme d'images, les algorithmes et capacités de calcul se sont fortement améliorées les dernières années, ce qui a ouvert la voie à de nouvelles approches.

Retour sur la génération d'images synthétiques

Nous avons expérimenté deux méthodes de génération d'images synthétiques. La première, que nous avons proposée, permet de créer des images RGB en projetant des modèles 3D sur des arrière-plans réels, et de leur ajouter des caractéristiques visuelles des images égocentriques. Nous avons montré que ces images synthétiques permettent d'avoir de meilleurs résultats que si l'on ne disposait que d'un petit nombre d'images réelles, car elles présentent beaucoup de variations dans les points de vue des objets. Nous n'avons cependant pas été jusqu'à simuler des occultations ni la présence de plusieurs objets par image, ce qui explique certaines erreurs de détection dans les images réelles. De plus, cette approche ne permet pas la génération d'images de profondeur associées aux images réelles. En revanche, de nombreux réseaux de neurones dédiés à la création de cartes de profondeur à partir d'une image RGB ont été proposés dans l'état de l'art (Atapour-Abarghouei et Breckon, 2018; Wofk *et al.*, 2019; Billy et Desbarats, 2020; Ranftl *et al.*, 2022). Ils pourraient être utilisés en complément du générateur d'images RGB, afin de produire des cartes de profondeur plus réalistes, incluant des défauts et dont les bords des objets sont moins nets.

La deuxième méthode que nous avons utilisée est la génération d'images entièrement synthétiques à partir de scènes 3D. Nous avons pour cela utilisé l'outil BlenderProc, qui permet de créer des scènes et d'en extraire des images RGB-D, éliminant ainsi le problème de cohérence entre les modalités. Malheureusement, les réseaux de neurones que nous avons entraînés avec ces images ont montré des difficultés à généraliser aux images de profondeur réelles. En améliorant l'apparence de celles-ci à l'inférence, nous avons réduit cet écart de domaine, en totalité pour le jeu de données LINEMOD, et partiellement pour le jeu de données T-LESS. Lorsque nous avons combiné les images de profondeur synthétiques avec les image RGB pour entraîner le réseau SSD multimodal, nous avons dépassé la performance obtenue avec chacune des modalités séparément, pour ces deux jeux de données. Pour RUSPA, en revanche, les images synthétiques générées à partir de scènes

3D semblent être trop différentes des images réelles, autant pour les images RGB que les images de profondeur. Ainsi, la fusion des modalités n'améliore pas la performance. Dans le cas des images 2D augmentées, l'utilisation d'un réseau YOLOv3 pré-entraîné sur les images réelles, plus proches des images de test, et dont les poids de l'extracteur de caractéristiques ont été figés, a permis d'améliorer la performance. Pour le réseau SSD uni-modal entraîné avec des images synthétiques issues de scènes 3D, figer les poids de MobileNetV3 pré-entraîné avec ImageNet n'a pas permis au réseau de converger, la performance ne s'améliorant pas au fur et à mesure de l'entraînement. Une piste d'amélioration serait de pré-entraîner le réseau multimodal avec des images réelles RGB-D, chaque branche séparément ou de façon conjointe, même si ces images proviennent d'un jeu de données non égocentrique tels que T-LESS ou LINEMOD.

Le développement des réseaux de transfert de domaine et de génération d'images a ouvert plusieurs pistes de réflexion quant à la façon de générer des données synthétiques et réalistes, ou de transférer les images synthétiques vers le domaine réel. Si des approches plutôt complexes avec de multiples transferts et réseaux ont été proposées (Liu *et al.*, 2019; Gu *et al.*, 2020), d'autres plus simples peuvent aussi être adoptées. Par exemple, des images de profondeur réelles issues de jeux de données publics pourraient servir de domaine cible afin de rendre les images synthétiques plus réalistes. C'est un peu l'idée derrière le réseau MiDAS de Ranftl *et al.* (2022), qui est entraîné avec des images RGB-D de 10 jeux de données différents afin de produire une image de profondeur, à partir d'une image RGB, qui s'adapte à différents capteurs. Nous avons d'ailleurs intégré ce réseau à différentes étapes de notre entraînement et inférence, mais sans obtenir d'amélioration de la performance du détecteur d'objets.

Dans la pratique, les méthodes qui s'appuient sur des images générées ou modifiées par un réseau de neurones sont très sensibles aux objets représentés dans les données d'entraînement, autant pour le transfert de domaine que pour la génération d'images. En effet, les méthodes de transfert de style ont tendance à modifier la forme ou la taille des objets présents dans les images source, ce qui peut fausser les annotations associées ; c'est un défaut rédhibitoire pour l'utilisation de ces réseaux dans le domaine médical par exemple (Cohen *et al.*, 2018). Nous l'avons nous-même expérimenté en entraînant le réseau de Park *et al.* (2020) pour le transfert de domaine et rendre plus réalistes les images synthétiques de profondeur du jeu de données T-LESS, en utilisant comme cible les images de profondeur réelles du jeu de données NYU Depth V2 (Silberman *et al.*, 2012). L'objectif était d'ajouter du bruit dans les images synthétiques, en particulier les pixels manquants sur les bords des objets ; le résultat général ressemble bien à ce qui était attendu, mais en regardant de plus près, les pixels manquants ont pris la forme de portes et de fenêtres, tels qu'elles apparaissent dans les images cibles. Les objets ont également été légèrement déformés, notamment en les allongeant verticalement comme s'il s'agissait de pieds de tables ou chaises. Ces images synthétiques modifiées par transfert de domaine ne nous ont pas permis l'amélioration des résultats du détecteur d'objets testé sur des images réelles. De plus, l'étude de Eilertsen *et al.* (2021) sur l'utilisation d'images générées par des GANs a montré qu'il était plus intéressant d'utiliser plusieurs GANs pour générer des images plus variées, chacun des réseaux se spécialisant sur un sous-ensemble des données, afin de reproduire plus fidèlement la distribution du jeu de données complet. De façon générale, ces méthodes nécessitent lors de l'entraînement, pour être efficaces, des images les plus proches possibles des images réelles auxquelles le réseau sera ensuite confronté, ce dont nous ne disposons pas dans notre contexte.

Au-delà de la détection d'objets

Dans cette thèse, nous avons travaillé principalement autour des images d'entraînement et de l'architecture du réseau de neurones pour la détection d'objets. Nous avons étudié deux architectures de réseaux de neurones *single stage*, YOLOv3 et SSD, et avons montré qu'elles permettent d'identifier des objets dans des images réelles avec uniquement des images synthétiques pendant l'entraînement. De plus, ces modèles sont suffisamment petits et rapides pour être utilisés dans le cadre d'une application embarquée de réalité augmentée. Nous avons également proposé une architecture multimodale basée sur SSD, qui permet d'améliorer la précision de la détection, pour une augmentation négligeable du nombre de paramètres.

Nous avons volontairement mis de côté la dimension temporelle des images à l'inférence, qui sont présentées au réseau sous forme de séquence vidéo. Alors que les objets sont tronqués dans les images lorsqu'ils sont manipulés, ils sont souvent visibles entièrement (ou presque) lorsqu'ils sont posés sur le plan de travail. Considérer les images comme une séquence, et non de façon indépendante, permettrait de capitaliser sur les détections effectuées dans les images précédentes, et ainsi faire moins d'erreurs de prédiction. Intégrer un algorithme de suivi temporel permet également de ne pas appliquer le détecteur à chaque image, ce qui réduit fortement le temps de calcul nécessaire par image ainsi que la consommation énergétique de l'algorithme. De nombreux algorithmes de suivi ont été proposés, qui s'intègrent dans des solutions d'apprentissage profond : SORT (Bewley *et al.*, 2016), mmMOT (Zhang *et al.*, 2019), ByteTrack (Zhang *et al.*, 2021a), etc.

Alors que nous nous sommes concentrés sur la détection d'objets avec un point de vue égocentrique et sans images réelles, il serait intéressant d'explorer les avantages d'autres tâches. Par exemple, la segmentation sémantique identifie de façon plus précise la géométrie des objets qu'une boîte englobante rectangulaire, ce qui pourrait faciliter l'identification de la catégorie. L'estimation de pose, bien que nécessitant plusieurs étapes, offre la possibilité de superposer les objets virtuels aux objets réels. Si l'annotation d'images réelles pour l'évaluation de ces approches est longue et fastidieuse, leur utilisation dans un contexte industriel est possible grâce aux images synthétiques, dont les annotations pour ces tâches sont créées aussi simplement que pour la détection d'objets.

Enfin, en entraînant YOLOv3 avec des images réelles, nous avons observé que le réseau apprenait particulièrement bien les positions relatives des objets lorsque le siège de bus est assemblé ou partiellement assemblé. Une autre direction de travail que nous n'avons pas explorée pourrait être de considérer le produit à assembler dans son ensemble, tel qu'un objet constitué de plusieurs parties. Plusieurs travaux proposent des approches variées à ce problème (Shen *et al.*, 2012; Aubry *et al.*, 2014; Crivellaro *et al.*, 2015). Détecter l'absence ou la présence des différentes parties d'un produit industriel permettrait également d'identifier automatiquement l'étape d'assemblage dans laquelle se trouve l'opérateur, et indiquer l'objet suivant à assembler.

Réduction des données nécessaires à l'inférence

Le développement de ces travaux de thèse correspondant à celui du prototype de casque de réalité augmentée de DEMS, nous avons aujourd'hui un peu plus de recul quant aux caractéristiques de l'ordinateur embarqué et des capacités de calcul. Le nombre de poids réduit du modèle SSD Lite permet amplement de le stocker en mémoire, ce qui nous a

encouragé à explorer une architecture plus conséquente pour améliorer la performance. Cependant, la consommation énergétique reste une limitation importante. En particulier, l'utilisation des deux modalités, la couleur et la profondeur, augmente la consommation du capteur au-delà de la limite, ce qui ne permet pas d'utiliser le système de façon autonome pendant les 4 heures requises. Il semble donc impossible pour l'instant d'utiliser l'image de profondeur lors du déploiement industriel du projet.

Nos expériences avec le détecteur d'objet multimodal ont cependant montré un avantage à combiner les caractéristiques extraites de plusieurs modalités. Si les images de profondeur ne peuvent pas être utilisées à l'inférence, elles pourraient tout de même être utilisées lors de l'entraînement ; la méthode « d'hallucination » de modalité proposée dans Hoffman *et al.* (2016) en est un exemple. Les travaux autour de la génération de modèles 3D à partir d'une image RGB (Wang *et al.*, 2018), ou encore de l'association de modèles 3D aux images RGB (Kuo *et al.*, 2020; Janik *et al.*, 2021), constituent également des pistes intéressantes. En effet, puisque les modèles des objets d'intérêt sont embarqués dans l'application, ils peuvent être utilisés également par l'algorithme de traitement d'images.

De nouvelles possibilités pour l'apprentissage profond

Si quelques plateformes existaient au début de cette thèse pour intégrer des réseaux de neurones dans des applications embarquées, leur nombre a augmenté les dernières années, ainsi que leur facilité d'utilisation. Ces plateformes permettent aujourd'hui de compresser les réseaux entraînés afin de les exécuter sur des supports spécifiques de façon plus rapide, y compris sans GPU. Ainsi, des réseaux de neurones de taille plus importante pourraient finalement être compressés et embarqués dans des applications mobiles pour une utilisation en temps réel, ce qui offrirait une meilleure capacité d'apprentissage. En particulier, nos expériences ont montré que les objets sont généralement bien localisés, mais mal identifiés par les « petits » modèles que nous avons utilisés. Il serait donc possible d'envisager un modèle en deux étapes, à l'image des détecteurs *double stage*, avec une étape de détection de région rapide et indépendante de la classe, afin de mettre l'accent sur l'identification des objets dans chaque région.

En ce qui concerne les architectures de réseaux, des extracteurs de caractéristiques plus performants voient encore le jour régulièrement. Ils semblent cependant arriver aux limites de leurs capacités, et différentes directions de recherche visent à les remplacer par de nouveaux paradigmes. En particulier, les réseaux de neurones « Transformers » appliqués aux images (appelés ViT pour Vision Transformers) semblent avoir une capacité d'apprentissage bien supérieure, aux dépens d'une taille toujours plus grande qui nécessite plus de capacité de calcul, plus de données d'entraînement, et un plus grand espace en mémoire (Lin *et al.*, 2021). Ils semblent également se prêter à la détection d'objets (Carion *et al.*, 2020), pour un temps d'inférence similaire à Faster R-CNN. L'architecture MobileViT (Mehta et Rastegari, 2021) a été proposée récemment dans le but de réduire la taille et la durée d'inférence de ces réseaux, afin de rendre possible leur utilisation dans des applications mobiles. Si ces nouvelles approches élargissent toujours plus les capacités des réseaux de neurones artificiels, il n'est pas encore clair si les ViT peuvent être utilisés sans images réelles. Nakashima *et al.* (2021) abordent la question dans un contexte de préservation de la vie privée, en utilisant un jeu de données dans lequel les objets sont générés comme des fractales, afin de ne pas utiliser les jeux de données habituels de pré-entraînement. Leurs résultats semblent montrer que les images générées de cette façon

sont très efficaces en combinaison avec les ViT, même s'il ne s'agit pas d'images classiques. Enfin, tout comme la préservation de la vie privée est un enjeu important pour le déploiement de solutions de traitement d'images à base d'apprentissage profond, l'impact écologique de ces solutions est également à prendre en compte. En particulier, l'entraînement d'un réseau de neurones, à travers l'utilisation de serveurs de calcul fonctionnant pendant des jours, voire des semaines, est aujourd'hui une préoccupation importante des chercheurs. Cela remet en question la nécessité de développer et utiliser des modèles massifs, et encourage les travaux de recherche dans une direction qui prend en compte leur efficacité énergétique.

Contributions au projet Katapuls

Pour conclure, nous avons contribué au projet Katapuls de DEMS en montrant la possibilité d'utiliser des images totalement synthétiques pour entraîner un réseau de neurones profond dans le cadre d'une application de réalité augmentée. Nous avons étudié différentes approches de génération d'images et différentes architectures neuronales, afin de proposer des lignes directrices pour le développement d'un algorithme d'analyse de l'environnement des opérateurs, depuis une caméra positionnée sur un casque de réalité augmentée. Les approches proposées ne nécessitent pas d'intervention d'un expert ni d'annotations manuelles, ce qui correspond au cahier des charges initial. L'application de ces travaux à d'autres produits industriels est en cours, avec des démonstrateurs réalisés en interne à l'entreprise.

Annexes

Sommaire

A BlenderProc

Cette annexe détaille les éléments principaux de la bibliothèque logicielle BlenderProc. Ces éléments constituent les blocs de base du fichier de configuration, à partir duquel il est possible de générer des jeux de données synthétiques de façon extrêmement simple. Il est à noter que BlenderProc est régulièrement mise à jour et améliorée, et a donc été modifiée depuis que nous l'avons utilisée. Nous décrivons ici son fonctionnement général, sans entrer dans les détails qui peuvent ne plus correspondre à son état actuel.

Fichier de configuration YAML

Tous les paramètres pour créer un jeu de données sont rassemblés dans un seul fichier YAML. Ce fichier est composé de plusieurs sections : la section « setup » définit les chemins d'accès nécessaires ; la section « global » définit les paramètres associés à tous les modules ; la section « modules » liste chacun des modules qui seront utilisés, et définit les paramètres de chacun d'entre eux.

En termes de programmation, chaque module correspond à une classe héritant d'une classe de base. A la lecture des modules dans le fichier de configuration, le script principal instancie des objets correspondant à chacun des modules. Une fois que tous les modules sont instanciés, la scène est générée avec les paramètres spécifiés, puis tous les modules sont exécutés successivement. Le dernier est le module d'écriture, qui enregistre les fichiers de sortie.

Module : *Loader*

Ce module charge tous les objets dans la scène. Ces objets peuvent correspondre aux objets d'intérêt, à des distracteurs, ou à l'environnement. Plusieurs types de *Loaders* sont définis par défaut dans BlenderProc : *BopLoaderModule* pour charger les objets du benchmark BOP (Hodan *et al.*, 2018) ; *Pix3DLoaderModule*, *IKEALoaderModule*, *SceneNetLoaderModule*, *ShapeNetLoaderModule* ou encore *SuncgLoaderModule* pour charger des objets ou scènes provenant respectivement des bases de données Pix3D (Sun *et al.*, 2018), IKEA (Lim *et al.*, 2013), SceneNet (McCormac *et al.*, 2017), ShapeNet (Chang *et al.*, 2015) et SunCG (Song *et al.*, 2017) ; *ObjectLoaderModule* pour charger n'importe quel modèle 3D. Des propriétés peuvent être associées à chacun des éléments chargés, tels que l'étiquette de classe ou si l'objet est soumis à la gravité.

Classe : *Provider*

Les objets de classe *Provider* ne sont pas des modules présents dans le fichier de configuration, mais des outils associés aux modules et permettant de manipuler les objets de la scène. Ils sont de deux types : *getter* et *sampler*. Un *getter* sélectionne des éléments de la scène correspondant à une liste de conditions, comme un type particulier (« MESH », « CAMERA », « LIGHT », etc.) ou une propriété. Les éléments sélectionnés sont ceux qui seront affectés par le module dans lequel le *getter* est inséré. Un *sampler* donne une valeur ou une liste de valeurs en fonction de son type. Par exemple, un *sampler* de type *Color* fournit les valeurs RGB correspondant à une couleur, alors qu'un *sampler* de type *Sphere* fournit une liste de points disposés sur la surface d'une sphère, en respectant des paramètres supplémentaires qui peuvent être spécifiés (intervalle de valeurs, rayon de la

sphère, etc.). Les valeurs données par le *sampler* sont attribuées aux propriétés des objets affectés par ce module, comme la couleur ou la valeur de réflexivité d'un matériau, la force d'une source lumineuse ou encore les positions successives d'une caméra.

Module : *Manipulators*

Ce module permet de modifier les objets chargés dans la scène. Il peut être utilisé pour appliquer des paramètres spécifiques aux propriétés de texture des objets (*MaterialManipulator*), ou pour modifier les objets eux-mêmes comme changer leur taille ou les rendre invisibles (*EntityManipulator*).

Module : *PhysicsPositioning*

Ce module gère la simulation des lois physiques dans la scène. Ses paramètres de configuration incluent la durée de la simulation, les valeurs de friction et de rebond, les collisions, etc. Il se charge de calculer et d'attribuer les nouvelles positions des objets pour chaque instant de la simulation.

Module : *ObjectPoseSampler*

Ce module génère des valeurs de position et d'orientation pour les objets sélectionnés. Il effectue des vérifications de collisions, et retente jusqu'à trouver un agencement satisfaisant, ou jusqu'à atteindre le paramètre *max_ iterations*.

Module : *Lighting*

Ce module définit la ou les sources lumineuses. Le type, la position, l'intensité ainsi que la couleur de chaque source peuvent être ajustés, ou échantillonnés à partir d'un *sampler*.

Module : *Camera*

Ce module spécifie tous les paramètres de la caméra : les positions auxquelles effectuer une acquisition, le nombre d'acquisitions, la distance minimale et maximale aux objets, la résolution et distance focale, etc. Ces éléments peuvent être inclus directement dans le fichier de configuration (*CameraSampler*), ou les positions peuvent être dans un fichier externe (*CameraLoader*).

Module : *Renderer*

Ce module gère le rendu de la scène en une ou plusieurs images, et propose différentes sorties : une image couleur (*RgbRenderer*), une image de flux optique entre des scènes spécifiées par l'utilisateur (*FlowRenderer*), et une carte de segmentation (*SegMapRenderer*). La carte de segmentation peut indiquer à la fois la catégorie des objets, ainsi qu'un identifiant unique pour chaque instance. Pour générer une carte de profondeur, les paramètres à spécifier se trouvent dans le générateur d'images RGB, la distance ne pouvant être générée sans créer l'image couleur associée.

Module : *Writer*

Le dernier module permet de générer les annotations associées à chaque image. Différents modules existent en fonction du format d'annotation souhaité et de la base de données (*BopWriter*, *CocoAnnotationsWriter*, *ShapeNetWriter*, etc.). Le module *Hdf5Writer* enregistre les annotations au format HDF5, qui est un format spécialisé dans les données scientifiques pour une manipulation efficace des grands volumes de données. BlenderProc propose également plusieurs fonctions de post-traitement.

B Images du jeu de données RUSPA

Cette annexe présente des images réelles et synthétiques associées au jeu de données RUSPA. Les images réelles RGB sont acquises avec plusieurs smartphones, les images RGB-D avec une caméra Intel RealSense D435i, et les images synthétiques sont générées avec les deux méthodes présentées dans le Chapitre 3. Nous rappelons les modèles 3D des objets à détecter ainsi que leur noms dans la Figure B.1.

(a) Siège de bus complet (b) Poignée (c) Dossier

(d) Profil droit (e) Profil gauche (f) Assise

FIGURE B.1 – Les modèles 3D associés au siège de bus RUSPA. (a) siège assemblé, (b)-(f) les différentes pièces à détecter.

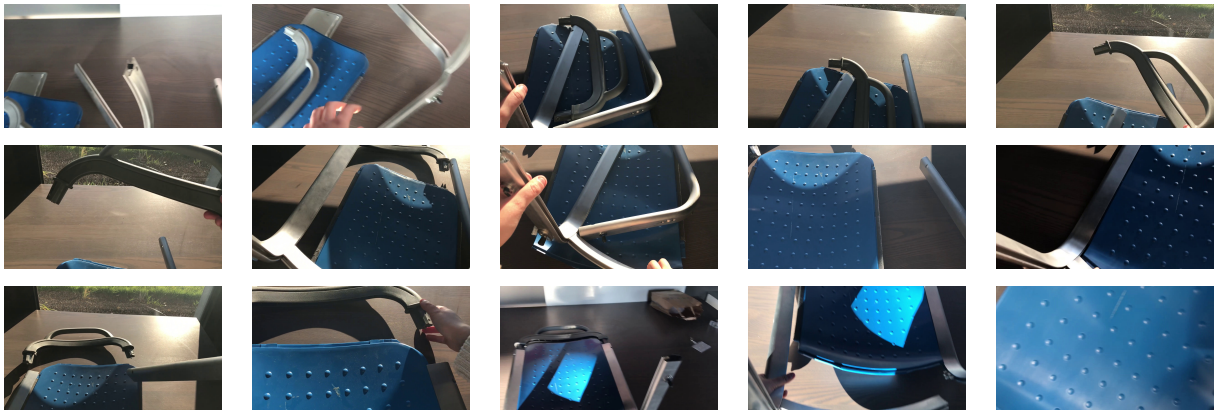
B.1 Images réelles

Les images présentées dans cette partie correspondent aux images couleur acquises en début de thèse avec différents smartphones (à la première et troisième personne), ainsi qu'aux paires d'images RGB-D acquises par la caméra Intel RealSense placée sur un prototype de casque de réalité augmentée. Les images RGB ont été utilisées à la fois pour

l'entraînement et la validation du modèle YOLOv3 ; les images RGB-D ont été utilisées uniquement comme ensemble de test pour le modèle SSD.

FIGURE B.2 – Photographies acquises avec un *smartphone* Honor 6C Pro pour l'entraînement et l'évaluation du modèles YOLOv3 (partie 4.2).

(a) Images acquises avec un smartphone Honor 6C Pro.



(b) Images acquises avec un smartphone iPhone 7 Plus.

FIGURE B.3 – Images issues de vidéos RGB égocentriques utilisées pour l'entraînement et l'évaluation du modèles YOLOv3 (partie 4.2).

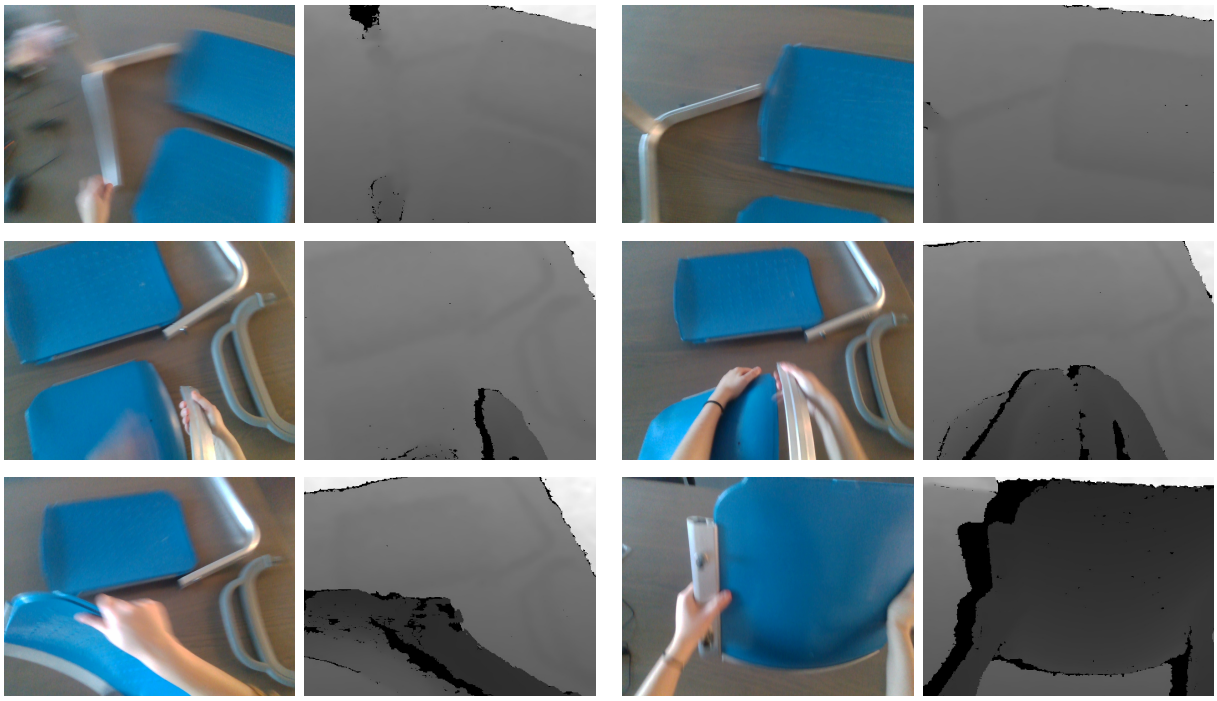


FIGURE B.4 – Images RGB-D extraites de vidéos égocentriques acquises par la caméra Intel RealSense D435i.

B.2 Images 2D augmentées RGB

Les images présentées ici ont été générées avec notre approche détaillée dans la partie 3.2. Elles sont composées de vues 2D de modèles 3D, dont la couleur est choisie de façon aléatoire, projetées sur une image réelle comme arrière-plan.

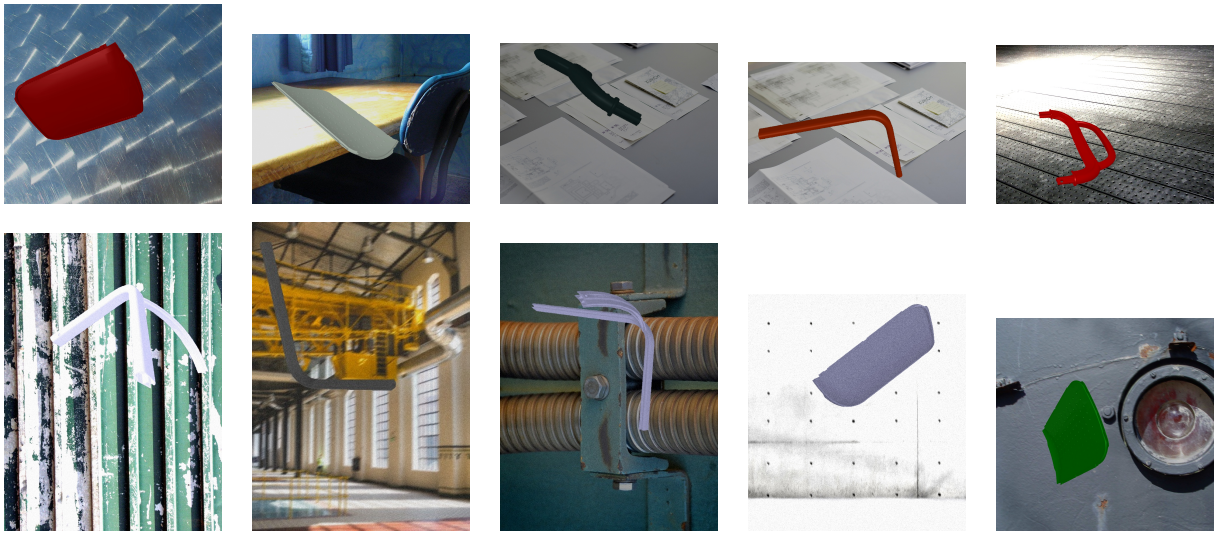


FIGURE B.5 – Images 2D augmentées synthétiques RGB du jeu de données RUSPA.

B.3 Images issues de scènes 3D RGB-D

Les images présentées ici ont été générées avec notre approche détaillée dans la partie 3.3. Chaque image RGB-D montre un ou plusieurs objets d'intérêt du jeu de données

RUSPA, et éventuellement un ou plusieurs objets distracteurs provenant d'un autre jeu de données.

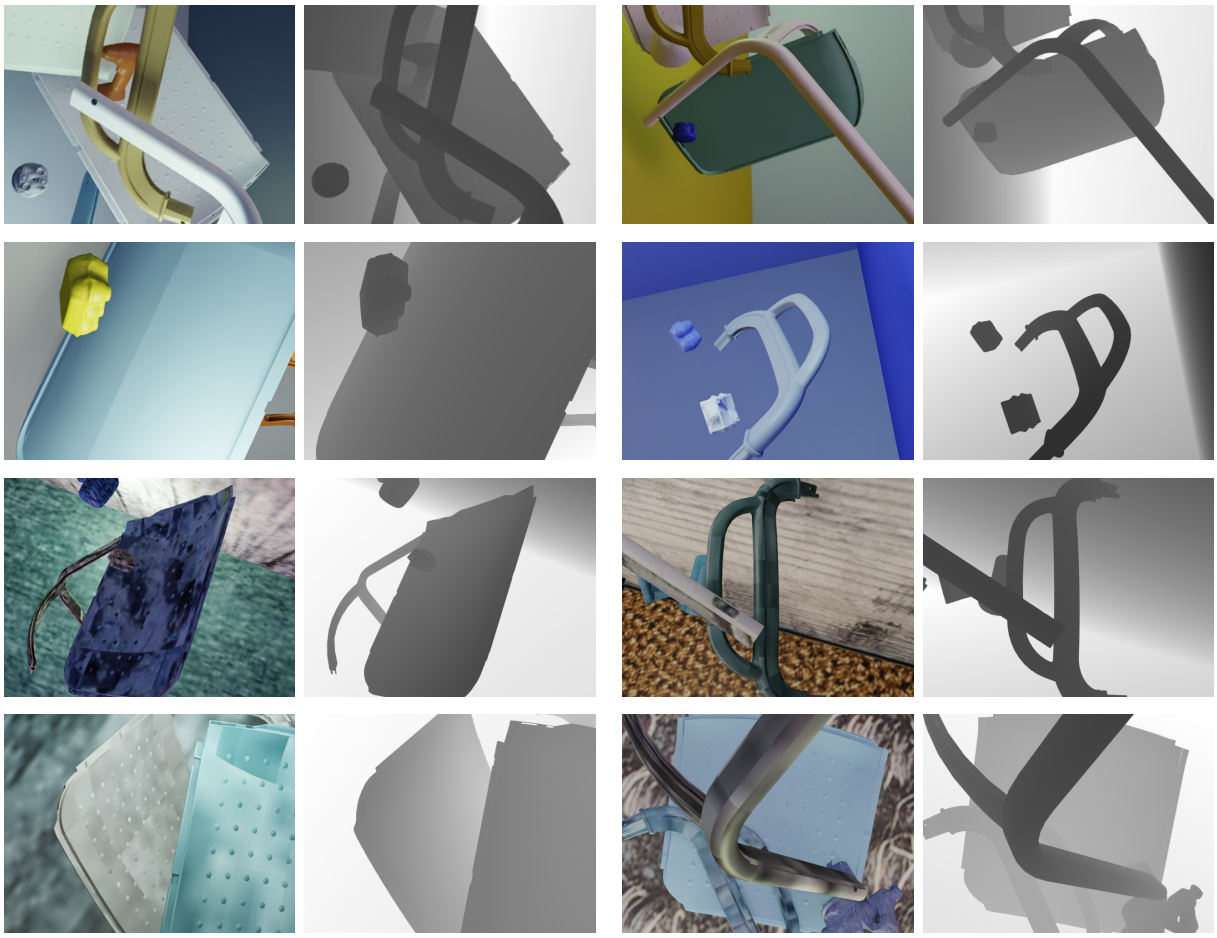


FIGURE B.6 – Exemples d'images synthétiques RGB-D du jeu de données RUSPA générées avec BlenderProc.

Bibliographie

Pulkit Agrawal, Joao Carreira et Jitendra Malik : Learning to see by moving. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 37–45, 2015.

Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars M. Mescheder, Andreas Geiger et Carsten Rother : Augmented reality meets computer vision : Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 126(9):961–972, 2018.

Amir Atapour-Abarghouei et Toby P. Breckon : Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2018.

Mathieu Aubry, Daniel Maturana, Alexei A. Efros, Bryan C. Russell et Josef Sivic : Seeing 3d chairs : Exemplar part-based 2d-3d alignment using a large dataset of CAD models. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2014.

Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang et Matthias NieBner : Scan2cad : Learning CAD model alignment in RGB-d scans. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Mehdi Ayadi, Leo Valque, Mihaela Scuturici, Serge Miguet et Chokri Ben Amar : The skyline as a marker for augmented reality in urban context. *In Advances in Visual Computing*, pages 698–711. Springer International Publishing, 2018.

Ronald T. Azuma : A survey of augmented reality. *Presence : Teleoperators and Virtual Environments*, 6(4):355–385, aug 1997.

Sven Bambach, Stefan Lee, David J. Crandall et Chen Yu : Lending a hand : Detecting hands and recognizing activities in complex egocentric interactions. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2015.

Peter Bartlett : For valid generalization the size of the weights is more important than the size of the network. *Advances in Neural Information Processing Systems*, 9, 1996.

Alona Baruhov et Guy Gilboa : Unsupervised enhancement of real-world depth images using tri-cycle gan. *arXiv preprint arXiv :2001.03779*, 2020.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert et Jason Weston : Curriculum learning. *In International Conference on Machine Learning (ICML)*. ACM Press, 2009.

Gedas Bertasius, Hyun Soo Park, Stella Yu et Jianbo Shi : First-person action-object detection with EgoNet. *In Robotics : Science and Systems XIII*. Robotics : Science and Systems Foundation, jul 2017.

Bassem Besbes, Sylvie Naudet Collette, Mohamed Tamaazousti, Steve Bourgeois et Vincent Gay-Bellile : An interactive augmented reality system : A prototype for industrial maintenance training applications. *In IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, nov 2012.

Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos et Ben Upcroft : Simple online and realtime tracking. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

Mark Billinghurst, Adrian Clark et Gun Lee : A survey of augmented reality. *Foundations and Trends® in Human–Computer Interaction*, 8(2-3):73–272, 2015.

Antoine Billy et Pascal Desbarats : DA-NET : Monocular depth estimation using disparity maps awareness NETwork. *In Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2020.

Jonas Blattgerste, Benjamin Strenge, Patrick Renner, Thies Pfeiffer et Kai Essig : Comparing conventional and augmented reality instructions for manual assembly tasks. *In Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*. ACM, jun 2017.

Gabriele Bleser, Yulian Pastarmov et Didier Stricker : Real-time 3d camera tracking for industrial augmented reality applications. 2005.

Marc Bolaños et Petia Radeva : Ego-object discovery. *arXiv preprint arXiv :1504.01639*, 2015.

Ali Borji, Ming-Ming Cheng, Huaizu Jiang et Jia Li : Salient object detection : A benchmark. *IEEE Transactions on Image Processing*, 24(12):5706–5722, dec 2015.

Steve Bourgeois, Boris Meden, Vincent Gay-Bellile, Mohamed Tamaazousti et Sebastian Knodel : Modeling, tracking, annotating and augmenting a 3d object in less than 5 minutes. *Vision and Content Engineering Laboratory*, 2016.

Behzad Bozorgtabar, Mohammad Saeed Rad, Dwarikanath Mahapatra et Jean-Philippe Thiran : Syndemo : Synergistic deep feature alignment for joint learning of depth and ego-motion. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4210–4219, 2019.

Sebastian Büttner, Michael Prilla et Carsten Röcker : Augmented reality training for industrial assembly work - are projection-based AR assistive systems an appropriate tool for assembly training ? *In CHI Conference on Human Factors in Computing Systems*. ACM, apr 2020.

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin et Alexandr A. Kalinin : Albumentations : Fast and flexible image augmentations. *Information*, 11(2):125, feb 2020.

Yuanzhouhan Cao, Chunhua Shen et Heng Tao Shen : Exploiting depth from single monocular images for object detection and semantic segmentation. *IEEE Transactions on Image Processing*, 26(2):836–846, feb 2017.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov et Sergey Zagoruyko : End-to-end object detection with transformers. *In Computer Vision – ECCV 2020*, pages 213–229. Springer International Publishing, 2020.

Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi et Fisher Yu : Shapenet : An information-rich 3d model repository. *arXiv preprint arXiv :1512.03012*, 2015.

R. Qi Charles, Hao Su, Mo Kaichun et Leonidas J. Guibas : PointNet : Deep learning on point sets for 3d classification and segmentation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jul 2017.

Yang Chen et Gérard Medioni : Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, apr 1992.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous et Yann LeCun : The loss surfaces of multilayer networks. *In Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman et Trevor Darrell : Best practices for fine-tuning visual classifiers to new domains. *In Proceedings of the IEEE European Conference on Computer Vision Workshops (ECCVW)*, pages 435–442. Springer International Publishing, 2016.

Joseph Paul Cohen, Margaux Luck et Sina Honari : Distribution matching losses can hallucinate features in medical image translation. *In Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 529–536. Springer International Publishing, 2018.

Julia Cohen, Carlos Crispim-Junior, Jean-Marc Chiappa et Laure Tougne : Training an embedded object detector for industrial settings without real images. *In 2021 IEEE International Conference on Image Processing (ICIP)*, pages 714–718. IEEE, 2021a.

Julia Cohen, Carlos Crispim-Junior, Céline Grange-Faivre et Laure Tougne : CAD-based learning for egocentric object detection in industrial context. *In 15th International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 5, pages 644–651. SCITEPRESS - Science and Technology Publications, 2020.

Julia Cohen, Carlos F Crispim-Junior, Jean-Marc Chiappa et Laure Tougne : MobileNet SSD : étude d'un détecteur d'objets embarquable entraîné sans images réelles. *In ORASIS 2021*, Saint Ferréol, France, septembre 2021b. Centre National de la Recherche Scientifique [CNRS]. URL `https://hal.archives-ouvertes.fr/hal-03531390`.

Guillaume Cortes, Eric Marchand, Guillaume Brincin et Anatole Lécuyer : MoSART : Mobile spatial augmented reality for 3d interaction with tangible objects. *Frontiers in Robotics and AI*, 5, aug 2018.

Alberto Crivellaro, Mahdi Rad, Yannick Verdie, Kwang Moo Yi, Pascal Fua et Vincent Lepetit : A novel representation of parts for accurate 3d object detection and tracking in monocular images. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2015.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan et Quoc V. Le : AutoAugment : Learning augmentation strategies from data. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens et Quoc V. Le : Randaugment : Practical automated data augmentation with a reduced search space. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, jun 2020.

Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price et Michael Wray : The EPIC-KITCHENS dataset : Collection, challenges and baselines. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(11):4125–4141, nov 2021.

Andrew J. Davison, Ian D. Reid, Nicholas D. Molton et Olivier Stasse : MonoSLAM : Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, jun 2007.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li et Li Fei-Fei : ImageNet : A large-scale hierarchical image database. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi et Harinandan Katam : Blenderproc. *arXiv preprint arXiv :1911.01911*, 2019.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun et Rob Fergus : Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

Terrance DeVries et Graham W Taylor : Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv :1708.04552*, 2017.

Clement Douarre, Carlos Crispim-Junior, Anthony Gelibert, Gerald Germain, Laure Tougne et David Rousseau : CTIS-net : A neural network architecture for compressed learning based on computed tomography imaging spectrometers. *IEEE Transactions on Computational Imaging*, 7:572–583, 2021.

Clément Douarre, Carlos F. Crispim-Junior, Anthony Gelibert, Laure Tougne et David Rousseau : On the value of CTIS imagery for neural-network-based classification : a simulation perspective. *Applied Optics*, 59(28):8697, sep 2020.

John Duchi, Elad Hazan et Yoram Singer : Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Debidatta Dwibedi, Ishan Misra et Martial Hebert : Cut, paste and learn : Surprisingly easy synthesis for instance detection. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.

Gabriel Eilertsen, Apostolia Tsirikoglou, Claes Lundström et Jonas Unger : Ensembles of gans for synthetic training data generation. *arXiv preprint arXiv :2104.11797*, 2021.

Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller et Wolfram Burgard : Multimodal deep learning for robust rgb-d object recognition. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687, 2015.

Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong et Ingmar Posner : Vote3deep : Fast object detection in 3d point clouds using efficient convolutional neural networks. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2017.

Dumitru Erhan, Aaron Courville, Yoshua Bengio et Pascal Vincent : Why does unsupervised pre-training help deep learning ? *In Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

Dumitru Erhan, Christian Szegedy, Alexander Toshev et Dragomir Anguelov : Scalable object detection using deep neural networks. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2014.

Gabriel Evans, Jack Miller, Mariangely Iglesias Pena, Anastacia MacAllister et Eliot Winer : Evaluating the Microsoft HoloLens through an augmented reality assembly application. *In Degraded environments : sensing, processing, and display 2017*, volume 10197, page 101970V. International Society for Optics and Photonics, 2017.

M. Everingham, L. Gool, C. K. Williams, J. Winn et A. Zisserman : The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

Deng-Ping Fan, Zheng Lin, Zhao Zhang, Menglong Zhu et Ming-Ming Cheng : Rethinking rgb-d salient object detection : Models, data sets, and large-scale benchmarks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5):2075–2089, 2021.

Alessandro Farasin, Francesco Peciarolo, Marco Grangetto, Elena Gianaria et Paolo Garza : Real-time object detection and tracking in mixed reality using microsoft HoloLens. *In International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*. SCITEPRESS - Science and Technology Publications, 2020.

Alireza Fathi, Ali Farhadi et James M. Rehg : Understanding egocentric activities. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, nov 2011a.

Alireza Fathi, Jessica K. Hodgins et James M. Rehg : Social interactions : A first-person perspective. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2012.

Alireza Fathi, Xiaofeng Ren et James M. Rehg : Learning to recognize objects in egocentric activities. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2011b.

Markus Funk, Andreas Bächler, Liane Bächler, Thomas Kosch, Thomas Heidenreich et Albrecht Schmidt : Working with augmented reality ? a long-term analysis of in-situ instructions at the assembly workplace. *In International Conference on Pervasive Technologies Related to Assistive Environments (PETRA)*, pages 222–229, 2017.

Markus Funk, Thomas Kosch, Scott W. Greenwald et Albrecht Schmidt : A benchmark for interactive augmented reality instructions for assembly tasks. *In Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*. ACM, nov 2015.

Antonino Furnari, Sebastiano Battiato, Kristen Grauman et Giovanni Maria Farinella : Next-active-object prediction from egocentric videos. *Journal of Visual Communication and Image Representation*, 49:401–411, nov 2017.

Adrien Gaidon, Qiao Wang, Yohann Cabon et Eleonora Vig : Virtualworlds as proxy for multi-object tracking analysis. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4340–4349, 2016.

Yaroslav Ganin et Victor Lempitsky : Unsupervised domain adaptation by backpropagation. *In International Conference on Machine Learning (ICML)*, pages 1180–1189, 2015.

Vincent Gay-Bellile, Steve Bourgeois, Mohamed Tamaazousti, Sylvie Naudet-Collette et Sebastian Knodel : A mobile markerless augmented reality system for the automotive field. *In ISMAR Workshop*, 2012.

Georgios Georgakis, Arsalan Mousavian, Alexander Berg et Jana Kosecka : Synthesizing training data for object detection in indoor scenes. *In Robotics : Science and Systems XIII*. Robotics : Science and Systems Foundation, jul 2017.

Ross Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik : Rich feature hierarchies for accurate object detection and semantic segmentation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

Ross Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik : Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(1):142–158, jan 2016.

Xavier Glorot et Yoshua Bengio : Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Christian Fuegen, Abrham Gebreselasie, Cristina Gonzalez, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jachym Kolar, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Murrell, Takumi Nishiyasu, Will Price, Paola Ruiz Puentes, Merey Ramazanova, Leda Sari, Kiran Somasundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Yunyi Zhu, Pablo Arbelaez, David Crandall, Dima Damen, Giovanni Maria Farinella, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan et Jitendra Malik : Ego4d : Around the World in 3,000 Hours of Egocentric Video. *CoRR*, abs/2110.07058, 2021. URL `https://arxiv.org/abs/2110.07058`.

Michael Gschwandtner, Roland Kwitt, Andreas Uhl et Wolfgang Pree : Blensor : blender sensor simulation toolbox. *In ISVC'11 Proceedings of the 7th international conference on Advances in visual computing - Volume Part II*, pages 199–208, 2011.

Xiao Gu, Yao Guo, Fani Deligianni et Guang-Zhong Yang : Coupled real-synthetic domain adaptation for real-world deep depth enhancement. *IEEE Transactions on Image Processing*, pages 1–1, 2020.

Saurabh Gupta, Pablo Arbelaez, Ross Girshick et Jitendra Malik : Aligning 3d models to RGB-d images of cluttered scenes. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

Saurabh Gupta, Ross B. Girshick, Pablo Andrés Arbeláez et Jitendra Malik : Learning rich features from rgb-d images for object detection and segmentation. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 345–360, 2014.

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe et Hideki Nakayama : Faster AutoAugment : Learning augmentation strategies using backpropagation. *In Computer Vision – ECCV 2020*, pages 1–16. Springer International Publishing, 2020.

K. He, G. Gkioxari, P. Dollar et R. Girshick : Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(2):386–397, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Delving deep into rectifiers : Surpassing human-level performance on ImageNet classification. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Zhenyi He et Xubo Yang : Hand-based interaction for object manipulation with augmented reality glasses. *In Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry - VRCAI '14*. ACM Press, 2014.

Vishakh Hegde et Reza Zadeh : Fusionnet : 3d object classification using multiple data representations. *arXiv preprint arXiv :1607.05695*, 2016.

Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer et Balaji Lakshminarayanan : Augmix : A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv :1912.02781*, 2019.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler et Sepp Hochreiter : Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Stefan Hinterstoisser, Selim Benhimane, Vincent Lepetit, Pascal Fua et Nassir Navab : Simultaneous recognition and homography extraction of local patches with a simple linear classifier. *In British Machine Vision Conference (BMVC)*, 2008.

Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart et Kurt Konolige : On pre-trained image features and synthetic images for deep learning. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 682–697, 2018.

Stefan Hinterstoisser, Olivier Pauly, Hauke Heibel, Martina Marek et Martin Bokeloh : An annotation saved is an annotation earned : Using fully synthetic training for object detection. *In Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2787–2796, 2019.

Toma Hodan, Dima Damen, Walterio Mayol-Cuevas et Jiri Matas : Efficient texture-less object detection for augmented reality guidance. *In IEEE International Symposium on Mixed and Augmented Reality Workshops (ISMARW)*. IEEE, sep 2015.

Tomas Hodan, Pavel Haluza, Stepan Obdrzalek, Jiri Matas, Manolis Lourakis et Xenophon Zabulis : T-less : An rgb-d dataset for 6d pose estimation of texture-less objects. *In IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888, 2017.

Tomas Hodan, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Fabian Manhardt, Federico Tombari, Tae-Kyun Kim, Jiri Matas et Carsten Rother : Bop : Benchmark for 6d object pose estimation. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 19–35, 2018.

Timon Hofer, Faranak Shamsafar, Nuri Benbarka et Andreas Zell : Object detection and autoencoder-based 6d pose estimation for highly cluttered bin picking. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021.

Judy Hoffman, Saurabh Gupta et Trevor Darrell : Learning with side information through modality hallucination. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2016.

Judy Hoffman, Saurabh Gupta, Jian Leong, Sergio Guadarrama et Trevor Darrell : Cross-modal adaptation for rgb-d detection. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5032–5039, 2016.

Saihui Hou, Zilei Wang et Feng Wu : Deeply exploit depth information for object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 19–27, 2016.

Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan et Yukun Zhu : Searching for mobilenetv3. *In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto et Hartwig Adam : Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun et Enhua Wu : Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(8):2011–2023, aug 2020.

Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan et Xiaojin Gong : PENet : Towards precise and efficient image guided depth completion. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2021.

Minyoung Huh, Pulkit Agrawal et Alexei A Efros : What makes imagenet good for transfer learning ? *arXiv preprint arXiv :1608.08614*, 2016.

Hiroshi Inoue : Data augmentation by pairing samples for images classification. *arXiv preprint arXiv :1801.02929*, 2018.

Sergey Ioffe et Christian Szegedy : Batch normalization : Accelerating deep network training by reducing internal covariate shift. *In International Conference on Machine Learning (ICML)*, pages 448–456. PMLR, 2015.

Mona Jalal, Josef Spjut, Ben Boudaoud et Margrit Betke : SIDOD : A synthetic image dataset for 3d object pose recognition with distractors. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 475–477, 2019.

Maciej Janik, Niklas Gard, Anna Hilsmann et Peter Eisert : Zero in on shape : A generic 2d-3d instance similarity metric learned from synthetic data. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2021.

Dinesh Jayaraman et Kristen Grauman : Learning image representations tied to egomotion. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2015.

Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen et Timo Aila : Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Tero KARRAS, Samuli LAINE et Timo AILA : A style-based generator architecture for generative adversarial networks. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Tero KARRAS, Samuli LAINE, Miika AITTALA, Janne HELLSTEN, Jaakko LEHTINEN et Timo AILA : Analyzing and improving the image quality of stylegan. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8110–8119, 2020.

Kenji KAWAGUCHI : Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.

Evangelos KAZAKOS, Jaesung HUH, Arsha NAGRANI, Andrew ZISSERMAN et Dima DAMEN : With a little help from my temporal context : Multimodal egocentric action recognition. *arXiv preprint arXiv :2111.01024*, 2021.

Wadim KEHL, Fabian MANHARDT, Federico TOMBARI, Slobodan ILIC et Nassir NAVAB : Ssd-6d : Making rgb-based 3d detection and 6d pose estimation great again. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1521–1529, 2017.

Wadim KEHL, Fausto MILLETARI, Federico TOMBARI, Slobodan ILIC et Nassir NAVAB : Deep learning of local RGB-d patches for 3d object detection and 6d pose estimation. *In Computer Vision – ECCV 2016*, pages 205–220. Springer International Publishing, 2016.

Jaekyum KIM, Junho KOH, Yecheol KIM, Jaehyung CHOI, Youngbae HWANG et Jun Won CHOI : Robust deep multi-modal learning based on gated information fusion network. *In Asian Conference on Computer Vision (ACCV)*, pages 90–106, 2018.

Diederik P KINGMA et Jimmy BA : Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

Georg KLEIN et David MURRAY : Parallel tracking and mapping for small AR workspaces. *In IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, nov 2007.

Georg KLEIN et David MURRAY : Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):369–380, 2009.

Mark A. KRAMER : Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, feb 1991.

Alex KRIZHEVSKY et Geoffrey E HINTON : Using very deep autoencoders for content-based image retrieval. *In ESANN*, volume 1, page 2. Citeseer, 2011.

Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON : Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Anders KROGH et John HERTZ : A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

Jason Ku, Ali Harakeh et Steven L. Waslander : In defense of classical image processing : Fast depth completion on the CPU. *In 15th Conference on Computer and Robot Vision (CRV)*. IEEE, 2018.

Abhijit Kundu, Yin Li et James M. Rehg : 3d-RCNN : Instance-level 3d object reconstruction via render-and-compare. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2018.

Weicheng Kuo, Anelia Angelova, Tsung-Yi Lin et Angela Dai : Mask2cad : 3d shape prediction by learning to segment and retrieve. *In Computer Vision – ECCV 2020*, pages 260–277. Springer International Publishing, 2020.

Yann Labbé, Justin Carpentier, Mathieu Aubry et Josef Sivic : CosyPose : Consistent multi-view multi-object 6d pose estimation. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 574–591, 2020.

Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang et Oscar Beijbom : PointPillars : Fast encoders for object detection from point clouds. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Julien Langlois, Harold Mouchère, Nicolas Normand et Christian Viard-Gaudin : 3d orientation estimation of industrial parts from 2d images using neural networks. *In Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods (ICPRAM)*, pages 409–416, 2018.

Rosario Leonardi, Francesco Ragusa, Antonino Furnari et Giovanni Maria Farinella : Egocentric human-object interaction detection exploiting synthetic data. *arXiv preprint arXiv :2204.07061*, 2022.

Anat Levin, Dani Lischinski et Yair Weiss : Colorization using optimization. *ACM Transactions on Graphics*, 23(3):689–694, 2004.

Cheng Li et Kris M. Kitani : Pixel-level hand detection in ego-centric videos. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2013.

Wenkai Li, A. Nee et S. Ong : A state-of-the-art review of augmented reality in engineering analysis and simulation. *Multimodal Technologies and Interaction*, 1(3):17, sep 2017.

Ming Liang, Bin Yang, Yun Chen, Rui Hu et Raquel Urtasun : Multi-task multi-sensor fusion for 3d object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Joseph J. Lim, Hamed Pirsiavash et Antonio Torralba : Parsing IKEA objects : Fine pose estimation. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2013.

Tianyang Lin, Yuxin Wang, Xiangyang Liu et Xipeng Qiu : A survey of transformers. *arXiv preprint arXiv :2106.04554*, 2021.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He et Piotr Dollar : Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(2):318–327, 2020.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár et C Lawrence Zitnick : Microsoft coco : Common objects in context. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

Keng-Chi Liu, Yi-Ting Shen, Jan Klopp et Liang-Gee Chen : What synthesis is missing : Depth adaptation integrated with weak supervision for indoor scene parsing. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2019.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu et Alexander C Berg : SSD : Single shot multibox detector. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

Mingsheng Long, Yue Cao, Jianmin Wang et Michael Jordan : Learning transferable features with deep adaptation networks. *In International Conference on Machine Learning (ICML)*, pages 97–105. PMLR, 2015.

Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio et Ken Goldberg : Dex-net 2.0 : Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *In Robotics : Science and Systems XIII*. Robotics : Science and Systems Foundation, jul 2017.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow et Brendan Frey : Adversarial autoencoders. *arXiv preprint arXiv :1511.05644*, 2015.

Federico Manuri et Andrea Sanna : A survey on applications of augmented reality. *ACSIJ Advances in Computer Science : an International Journal*, 5(1):18–27, 2016.

Eric Marchand, Hideaki Uchiyama et Fabien Spindler : Pose estimation for augmented reality : A hands-on survey. *IEEE Transactions on Visualization and Computer Graphics*, 22(12):2633–2651, dec 2016.

Jonathan Masci, Ueli Meier, Dan Cireşan et Jürgen Schmidhuber : Stacked convolutional auto-encoders for hierarchical feature extraction. *In Lecture Notes in Computer Science*, pages 52–59. Springer Berlin Heidelberg, 2011.

Francisco Massa, Bryan C Russell et Mathieu Aubry : Deep exemplar 2d-3d detection by adapting from real to rendered views. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6024–6033, 2016.

Daniel Maturana et Sebastian Scherer : VoxNet : A 3d convolutional neural network for real-time object recognition. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015.

John McCormac, Ankur Handa, Stefan Leutenegger et Andrew J. Davison : SceneNet RGB-d : Can 5m synthetic images beat generic ImageNet pre-training on indoor

segmentation ? *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.

Sachin Mehta et Mohammad Rastegari : Mobilevit : light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv :2110.02178*, 2021.

Jean-Philippe Mercier, Mathieu Garon, Philippe Giguère et Jean-François Lalonde : Learning to match templates for unseen instance detection, 2019.

Daniel Mas Montserrat, Qian Lin, Jan Allebach et Edward J. Delp : Training object detection and recognition CNN models using data augmentation. *Electronic Imaging*, 29(10):27–36, jan 2017.

Raul Mur-Artal, J. M. M. Montiel et Juan D. Tardos : ORB-SLAM : A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015.

Vinod Nair et Geoffrey E Hinton : Rectified linear units improve restricted boltzmann machines. *In International Conference on Machine Learning (ICML)*, 2010.

Kodai Nakashima, Hirokatsu Kataoka, Asato Matsumoto, Kenji Iwata et Nakamasa Inoue : Can vision transformers learn without natural images ? *arXiv preprint arXiv :2103.13023*, 2021.

Natalia Neverova, Christian Wolf, Graham Taylor et Florian Nebout : Moddrop : Adaptive multi-modal gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(8):1692–1706, 2016.

Sergey I. Nikolenko : *Synthetic Data for Deep Learning*. Springer International Publishing, 2021.

Fernando Camaro Nogues, Andrew Huie et Sakyasingha Dasgupta : Object detection using domain randomization and generative adversarial refinement of synthetic images. *arXiv preprint arXiv :1805.11778*, 2018.

Viktor Olsson, Wilhelm Tranheden, Juliano Pinto et Lennart Svensson : Classmix : Segmentation-based data augmentation for semi-supervised learning. *In IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1369–1378, 2021.

Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan et Joseph Walsh : Deep learning vs. traditional computer vision. *In Advances in Intelligent Systems and Computing*, pages 128–144. Springer International Publishing, apr 2019.

Tanguy Ophoff, Kristof Van Beeck et Toon Goedemé : Exploring rgb+depth fusion for real-time object detection. *Sensors*, 19(4):866, 2019.

Rafael Padilla, Sergio L. Netto et Eduardo A. B. da Silva : A survey on performance metrics for object-detection algorithms. *In Proceedings of the 2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020.

Kiru Park, Timothy Patten et Markus Vincze : Pix2pose : Pixel-wise coordinate regression of objects for 6d pose estimation. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2019.

Sebeom Park, Shokhrukh Bokijonov et Yosoon Choi : Review of microsoft HoloLens applications over the past five years. *Applied Sciences*, 11(16):7259, aug 2021.

Taesung Park, Alexei A. Efros, Richard Zhang et Jun-Yan Zhu : Contrastive learning for unpaired image-to-image translation. *In Computer Vision – ECCV 2020*, pages 319–345. Springer International Publishing, 2020.

Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G. Derpanis et Kostas Daniilidis : 6-dof object pose from semantic keypoints. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2011–2018, 2017.

Xingchao Peng, Baochen Sun, Karim Ali et Kate Saenko : Exploring invariances in deep convolutional neural networks using synthetic images. *arXiv : Computer Vision and Pattern Recognition*, 2014.

Xingchao Peng, Baochen Sun, Karim Ali et Kate Saenko : Learning deep object detectors from 3d models. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1278–1286, 2015.

Benjamin Planche, Ziyan Wu, Kai Ma, Shanhui Sun, Stefan Kluckner, Oliver Lehmann, Terrence Chen, Andreas Hutter, Sergey Zakharov, Harald Kosch et Jan Ernst : Depthsynth : Real-time realistic synthetic data generation from cad models for 2.5d recognition. *In 2017 International Conference on 3D Vision (3DV)*, pages 1–10, 2017.

Yair Poleg, Chetan Arora et Shmuel Peleg : Temporal segmentation of egocentric videos. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2014.

Lucia Pombo et Margarida Morais Marques : Marker-based augmented reality application for mobile learning in an urban park : Steps to make it real under the EduPARK project. *In International Symposium on Computers in Education (SIIE)*. IEEE, nov 2017.

Ning Qian : On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

Mahdi Rad et Vincent Lepetit : BB8 : A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.

Mahdi Rad, Markus Oberweger et Vincent Lepetit : Domain transfer for 3d pose estimation from color images without manual annotations. *In Computer Vision – ACCV 2018*, pages 69–84. Springer International Publishing, 2019.

Francesco Ragusa, Antonino Furnari, Salvatore Livatino et Giovanni Maria Farinella : The MECCANO dataset : Understanding human-object interactions from egocentric videos in an industrial-like domain. *In 2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2021.

Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler et Vladlen Koltun : Towards robust monocular depth estimation : Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(3):1623–1637, mar 2022.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan et Stefan Carlsson : CNN features off-the-shelf : An astounding baseline for recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 512–519, 2014.

J. Redmon, A. Bochkovskiy et S. Sinigardi : Darknet : Yolov3 - neural network for object detection. `https://github.com/AlexeyAB/darknet`, 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi : You only look once : Unified, real-time object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

Joseph Redmon et Ali Farhadi : YOLO9000 : Better, faster, stronger. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Joseph Redmon et Ali Farhadi : Yolov3 : An incremental improvement. *arXiv preprint arXiv :1804.02767*, 2018.

S. Ren, K. He, R. Girshick et J. Sun : Faster R-CNN : Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

Gernot Riegler, Ali Osman Ulusoy et Andreas Geiger : OctNet : Learning deep 3d representations at high resolutions. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jul 2017.

Alina Roitberg, Tim Pollert, Monica Haurilet, Manuel Martin et Rainer Stiefelhagen : Analysis of deep fusion strategies for multi-modal gesture recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 198–206, 2019.

Frank Rosenblatt : The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Ross Girshick : Fast R-CNN. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

Avraham Ruderman, Neil C Rabinowitz, Ari S Morcos et Daniel Zoran : Pooling is neither necessary nor sufficient for appropriate deformation stability in cnns. *arXiv preprint arXiv :1804.04438*, 2018.

David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams : Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986.

Martin Runz, Maud Buffier et Lourdes Agapito : MaskFusion : Real-time recognition, tracking and reconstruction of multiple moving objects. *In IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, oct 2018.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford et Xi Chen : Improved techniques for training gans. *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 29, 2016.

Igor Sampaio, Luigy Machaca, José Viterbo et Joris Guérin : A novel method for object detection using deep learning and CAD models. *In Proceedings of the 23rd International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications, 2021.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov et L.-C. Chen : MobileNetV2 : Inverted residuals and linear bottlenecks. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

Kripasindhu Sarkar, Kiran Varanasi et Didier Stricker : Trained 3d models for CNN based object recognition. *In International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 130–137, 2017.

Max Schwarz, Anton Milan, Arul Selvam Periyasamy et Sven Behnke : Rgb-d object detection and semantic segmentation for autonomous manipulation in clutter. *The International Journal of Robotics Research*, 37:437–451, 2018.

Chao-Hui Shen, Hongbo Fu, Kang Chen et Shi-Min Hu : Structure recovery by part assembly. *ACM Transactions on Graphics*, 31(6):1–11, nov 2012.

Shreyas S. Shivakumar, Ty Nguyen, Ian D. Miller, Steven W. Chen, Vijay Kumar et Camillo J. Taylor : DFuseNet : Deep fusion of RGB and sparse depth information for image guided dense depth completion. *In IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, oct 2019.

Connor Shorten et Taghi M. Khoshgoftaar : A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), jul 2019.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli et Rob Fergus : Indoor segmentation and support inference from rgbd images. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 746–760, 2012.

Martin Simon, Stefan Milz, Karl Amende et Horst-Michael Gross : Complex-YOLO : An euler-region-proposal for real-time 3d object detection on point clouds. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 197–209. Springer International Publishing, 2019.

Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying et Quoc V Le : Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv :1711.00489*, 2017.

Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva et Thomas Funkhouser : Semantic scene completion from a single depth image. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 190–198, 2017.

Emiliano Spera, Antonino Furnari, Sebastiano Battiato et Giovanni Maria Farinella : Egocentric shopping cart localization. *In International Conference on Pattern Recognition (ICPR)*. IEEE, aug 2018.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox et Martin Riedmiller : Striving for simplicity : The all convolutional net. *arXiv preprint arXiv :1412.6806*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever et Ruslan Salakhutdinov : Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Hao Su, Charles R Qi, Yangyan Li et Leonidas J Guibas : Render for CNN : Viewpoint estimation in images using cnns trained with rendered 3d model views. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2686–2694, 2015.

Cecilia Summers et Michael J. Dinneen : Improved mixed-example data augmentation. *In IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2019.

Baochen Sun et Kate Saenko : From virtual to reality : Fast adaptation of virtual object detectors to real domains. *In Proceedings of the British Machine Vision Conference 2014*. British Machine Vision Association, 2014.

Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B. Tenenbaum et William T. Freeman : Pix3d : Dataset and methods for single-image 3d shape modeling. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2974–2983, 2018.

Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker et Rudolph Triebel : Implicit 3d orientation learning for 6d object detection from rgb images. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 712–729, 2018.

Chris Sweeney, Greg Izatt et Russ Tedrake : A supervised approach to predicting noise in depth images. *In International Conference on Robotics and Automation (ICRA)*, pages 796–802, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke et Andrew Rabinovich : Going deeper with convolutions. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

Ryo Takahashi, Takashi Matsubara et Kuniaki Uehara : Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, 2019.

Takafumi Taketomi, Hideaki Uchiyama et Sei Ikeda : Visual SLAM algorithms : a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9 (1), jun 2017.

Mohamed Tamaazousti, Vincent Gay-Bellile, Sylvie Naudet Collette, Steve Bourgeois et Michel Dhome : NonLinear refinement of structure from motion reconstruction by taking advantage of a partial knowledge of the environment. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2011.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard et Quoc V. Le : MnasNet : Platform-aware neural architecture search for mobile. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

Jie Tang, Fei-Peng Tian, Wei Feng, Jian Li et Ping Tan : Learning guided convolutional network for depth completion. *IEEE Transactions on Image Processing*, 30:1116–1129, 2021.

Yansong Tang, Yi Tian, Jiwen Lu, Jianjiang Feng et Jie Zhou : Action recognition in RGB-d egocentric videos. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2017.

Yansong Tang, Zian Wang, Jiwen Lu, Jianjiang Feng et Jie Zhou : Multi-stream deep neural networks for RGB-d egocentric action recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(10):3001–3015, oct 2019.

Maxim Tatarchenko, Alexey Dosovitskiy et Thomas Brox : Octree generating networks : Efficient convolutional architectures for high-resolution 3d outputs. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.

Hooman Tavakoli, Snehal Walunj, Parsha Pahlevannejad, Christiane Plociennik et Martin Ruskowski : Small object detection for near real-time egocentric perception in a manual assembly scenario. *arXiv preprint arXiv :2106.06403*, 2021.

Bugra Tekin, Sudipta N. Sinha et Pascal Fua : Real-time seamless single shot 6d object pose prediction. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2018.

Stefan Thalhammer, Kiru Park, Timothy Patten, Markus Vincze et Walter Kropatsch : Sydd : Synthetic depth data randomization for object detection using domain-relevant background. *TUGraz OPEN Library*, pages 14–22, 2019.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba et Pieter Abbeel : Domain randomization for transferring deep neural networks from simulation to the real world. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon et Stan Birchfield : Training deep networks with synthetic data : Bridging the reality gap by domain randomization. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 969–977, 2018.

Satoshi Tsutsui, Ruta Desai et Karl Ridgeway : How you move your head tells what you do : Self-supervised video representation learning with egocentric cameras and imu sensors. *arXiv preprint arXiv :2110.01680*, 2021.

J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers et A. W. M. Smeulders : Selective search for object recognition. *International Journal of Computer Vision (IJCV)*, 104(2):154–171, apr 2013.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio et Pierre-Antoine Manzagol : Extracting and composing robust features with denoising autoencoders. *In Proceedings of the 25th international conference on Machine learning - ICML '08*. ACM Press, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol et Léon Bottou : Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu et Yu-Gang Jiang : Pixel2mesh : Generating 3d mesh models from single RGB images. *In Computer Vision – ECCV 2018*, pages 55–71. Springer International Publishing, 2018.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun et Xin Tong : O-cnn : octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics*, 36(4):72, 2017.

Peng-Shuai Wang, Chun-Yu Sun, Yang Liu et Xin Tong : Adaptive O-CNN : a patch-based deep representation of 3d shapes. *ACM Transactions on Graphics*, 37(6):217, 2019.

Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman et Vivienne Sze : FastDepth : Fast monocular depth estimation on embedded systems. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2019.

Paul Wohlhart et Vincent Lepetit : Learning descriptors for object recognition and 3d pose estimation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3109–3118, 2015.

Nicolai Wojke, Alex Bewley et Dietrich Paulus : Simple online and realtime tracking with a deep association metric. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2017.

Daniel Worrall et Gabriel Brostow : CubeNet : Equivariance to 3d rotation and translation. *In Computer Vision – ECCV 2018*, pages 585–602. Springer International Publishing, 2018.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang et Jianxiong Xiao : 3d shapenets : A deep representation for volumetric shapes. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

Yu Xiang, Roozbeh Mottaghi et Silvio Savarese : Beyond PASCAL : A benchmark for 3d object detection in the wild. *In IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 75–82, 2014.

Christopher Xie, Yu Xiang, Arsalan Mousavian et Dieter Fox : The best of both modes : Separately leveraging rgb and depth for unseen object instance segmentation. *In Conference on Robot Learning*, pages 1369–1378. PMLR, 2020.

Xiangyang Xu, Yuncheng Li, Gangshan Wu et Jiebo Luo : Multi-modal deep feature learning for rgb-d object detection. *Pattern Recognition*, 72:300–313, 2017.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze et Hartwig Adam : NetAdapt : Platform-aware neural network adaptation for mobile applications. *In Computer Vision – ECCV 2018*, pages 289–304. Springer International Publishing, 2018.

Fisher Yu et Vladlen Koltun : Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv :1511.07122*, 2015.

Ye Yuan et Kris Kitani : 3d ego-pose estimation via imitation learning. *In Computer Vision – ECCV 2018*, pages 763–778. Springer International Publishing, 2018.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo et Junsuk Choe : CutMix : Regularization strategy to train strong classifiers with localizable features. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2019.

Heng Zhang, Elisa Fromont, Sebastien Lefevre et Bruno Avignon : Multispectral fusion for object detection with cyclic fuse-and-refine blocks. *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, oct 2020.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin et David Lopez-Paz : mixup : Beyond empirical risk minimization. *arXiv preprint arXiv :1710.09412*, 2017.

Wenwei Zhang, Hui Zhou, Shuyang Sun, Zhe Wang, Jianping Shi et Chen Change Loy : Robust multi-modality multi-object tracking. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2019.

Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu et Xinggang Wang : Bytetrack : Multi-object tracking by associating every detection box. *arXiv preprint arXiv :2110.06864*, 2021a.

Yinda Zhang et Thomas Funkhouser : Deep depth completion of a single RGB-d image. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2018.

Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba et Sanja Fidler : DatasetGAN : Efficient labeled data factory with minimal human effort. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2021b.

Zhengyou Zhang : Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision (IJCV)*, 13(2):119–152, oct 1994.

Xiaoqi Zhao, Youwei Pang, Lihe Zhang, Huchuan Lu et Xiang Ruan : Self-supervised representation learning for rgb-d salient object detection. *arXiv preprint arXiv :2101.12482*, 2021.

Xiaoqi Zhao, Lihe Zhang, Youwei Pang, Huchuan Lu et Lei Zhang : A single stream network for robust and real-time rgb-d salient object detection. *In Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 646–662. Springer, 2020.

Liang Zheng, Yi Yang et Qi Tian : SIFT meets CNN : A decade survey of instance retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(5):1224–1244, may 2018.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li et Yi Yang : Random erasing data augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (07):13001–13008, apr 2020.

Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens et Quoc V. Le : Learning data augmentation strategies for object detection. *In Computer Vision – ECCV 2020*, pages 566–583. Springer International Publishing, 2020.

Liste des figures

Liste des tableaux