



HAL
open science

Détection 3D pour la réalité mixte en maintenance industrielle

Philippe Pérez de San Roman

► **To cite this version:**

Philippe Pérez de San Roman. Détection 3D pour la réalité mixte en maintenance industrielle. Traitement des images [eess.IV]. Université de Bordeaux, 2022. Français. NNT : 2022BORD0419 . tel-04089831

HAL Id: tel-04089831

<https://theses.hal.science/tel-04089831v1>

Submitted on 5 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE :

DOCTEUR DE
L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUE ET D'INFORMATIQUE
SPÉCIALITÉ INFORMATIQUE

Par Philippe PÉREZ DE SAN ROMAN

DÉTECTION 3D
POUR LA RÉALITÉ MIXTE
EN MAINTENANCE INDUSTRIELLE

Sous la direction de :

M. Pascal DESBARATS, Professeur des universités, Université de Bordeaux & LaBRI, Directeur de thèse.
M. Jean-Philippe DOMENGER, Professeur des universités, Université de Bordeaux & LaBRI, Co-Directeur de thèse.

Soutenue le 9 Décembre 2022

Membre du jury :

M. Bertrand KERAUTRET,	Professeur des universités,	Université Lumière Lyon 2 & LIRIS,	Rapporteur & Président du jury.
M. Pascal BALLEZ,	Maitre de conférences,	Université de Brest & LATIM,	Rapporteur.
M. Pascal DESBARATS,	Professeur des universités,	Université de Bordeaux & LaBRI,	Directeur de thèse.
M. Jean-Philippe DOMENGER,	Professeur des universités,	Université de Bordeaux & LaBRI,	Co-Directeur de thèse.
M. Axel BUENDIA,	Professeur des universités,	Enjmin & CNAM-CEDRIC,	Invité.
M. Arnaud FAVAREILLE	Directeur Général,	ITECAC,	Invité.

Résumé

ITECA est développeur du logiciel *WITTY3D* qui est solution d'aide à la maintenance de machines industrielles. Une IA aide au diagnostic et à la résolution des pannes. Les pannes sont affichées sur le modèle 3D de la machine et des animations illustrent la procédure de résolution. Ce logiciel est aujourd'hui déployé sur PC fixes, PC portables, tablettes et smart-phones. Comparé à des documents textes et des schémas 2D, l'affichage 3D permet aux techniciens de plus facilement interpréter les informations affichées même si le problème ne disparaît pas complètement. En effet, ils ont toujours besoin d'interpréter et transposer les informations virtuelles qui leurs sont présentées à la réalité de terrain. Or, toute interprétation requiert un effort cognitif qui est source d'erreurs. Pour outrepasser ces limitations, ITECA souhaite faire évoluer *WITTY3D* en le portant sur des casques de réalités mixtes comme le Hololens de Microsoft. Ces casques utilisent des écrans transparents pour afficher des images directement devant les yeux de l'utilisateur. L'effet de perspective est alors parfait et permet de calquer les visuels virtuels sur l'environnement réel qui reste visible à travers la visière du casque. Ces casques fournissent le matériel d'affichage mais c'est au développeur de se doter des outils logiciels pour recalculer correctement les visuels. Plus précisément, le logiciel doit inférer la position et l'orientation des objets d'intérêts en 3D par rapport au casque.

Ce problème est connu dans la littérature comme celui de l'estimation des six degrés de liberté d'un objet (estimation des 6DoF). Une solution à ce problème consiste à utiliser des réseaux de neurones pour identifier, localiser dans l'image, et localiser en 3D, l'objet d'intérêt.

Dans cette thèse, nous avons considéré AlexNet, VGG-19 et GoogLeNet pour la classification d'images, YOLO pour la localisation dans l'image, et PoseNet et BB8 pour l'estimation des 6DoF. Nous les avons évalués sur trois jeux de données de l'état de l'art pour l'estimation des 6DoF : LINEMOD, T-LESS et YCB-VIDÉO. Les résultats de classifications d'images d'AlexNet, VGG-19 et GoogLeNet sont très bons sur ces jeux de données, qui proposent moins de catégories d'objets et moins de facteurs de difficultés, sauf sur T-LESS qui ne proposent pas assez d'images d'entraînement et dont les étiquettes sont ambiguës à cause des sous-parties d'objets communes à plusieurs objets. Nous n'avons pas approfondi la localisation d'objets dans l'image. Bien que nous ayons obtenu des résultats intéressants sur LINEMOD, les résultats sont peu concluants sur YCB-VIDÉO, et nous ne l'avons pas testé sur T-LESS par manque d'images d'entraînement. Pour l'estimation des 6DoF avec PoseNet et BB8 nous avons souhaité entraîner et tester les réseaux sur les mêmes images pour faire ressortir les différences dans le choix de l'architecture et de la modélisation de la pose utilisée (3D/2D). Les résultats sont prometteurs sur LINEMOD, surtout pour BB8 qui, sans augmentations d'images avancées, recalcule correctement les objets en 2D dans 23% des images de tests. Mais pour PoseNet, ou sur les autres jeux de données, nous avons rencontré des problèmes de sur-spécialisation. Ces problèmes sont dus au fait que ces trois jeux de données reposent sur l'utilisation de plus d'augmentations d'images qu'il ne nous en ait permis avec PoseNet, ou sur l'utilisation de rendus 3D photoréalistiques comme images de substitution.

Les sur-spécialisations observées sont dues au fait que deux des jeux de données considérés ne sont pas conçus pour entraîner des réseaux de neurones, et le troisième est destiné à des applications de préhension robotique, ce qui n'est pas notre objectif. Plusieurs problèmes de qualité des étiquettes sont aussi apparus et, de façon générale, ces jeux de données proposent des points de vue distants ou mal distribués. Tout cela ne permet pas d'obtenir des résultats d'estimations de 6DoF satisfaisants, même avec un seuil de précision de 5° , 5 cm ou 5 px . Or, nous souhaitons obtenir des résultats meilleurs, et ce, avec des seuils de précisions plus stricts et plus précis.

Pour outrepasser ces problèmes nous avons décidé de constituer notre propre jeu de données, appelé NEMA. Comme LINEMOD et T-LESS nous utilisons des marqueurs pour obtenir des étiquettes de qualité. Mais, contrairement à ces derniers, notre jeu de données fournit les arrières-plans qui permettent d'effacer ces marqueurs. Nous avons aussi photographié les images avec un pas plus fin, ce qui nous permet d'obtenir plus d'images, et des points de vue très denses. De ce fait, nous pouvons entraîner sans problèmes PoseNet et BB8 à des précisions jamais atteintes auparavant. Quand les méthodes étaient évaluées à des précisions de 5 cm , 5° ou 5 px , nous les évaluons, nous à 2 cm , 2° , 2 px ou moins avec de très bon résultats. À 2 px près, les recalages 2D proposés par BB8 sont corrects en moyenne dans 96.80% des images de tests, et dans 28.24% à 1 px . Et à 2° et 2 cm , les paramètres de poses sont corrects dans 75.40% des images. PoseNet est moins précis en 2D avec seulement 12.21% des images correctement recalées à 2 px près. Ces résultats montrent que les performances de PoseNet et BB8 sur le jeu de données de l'état de l'art étaient limités par les données d'apprentissage. Nous avons aussi comparé chaque méthode avec elle-même, une fois avec arrières-plans aléatoires, et une fois avec les arrières-plans réels. Ces expériences ont montré que les arrières-plans aléatoires limitent la sur-spécialisation du réseau, ce qui est une bonne chose dans le cas de PoseNet. Mais, dans le cas de BB8 cela prévient sa bonne spécialisation sur NEMA et impacte négativement les résultats : les résultats de recalage 2D sont 8.51% meilleurs, à 1 px près, avec les arrières plans réels.

Cette thèse commence par une courte introduction du sujet, du cadre en entreprise et des contributions. La suite est décomposée en 4 parties :

- La première partie détaille le sujet de la thèse et la problématique. Nous apporterons un certain nombre de définitions et expliquerons nos inspirations ainsi que les retours d'expériences de nos partenaires industriels. Nous y présenterons les supports matériels et illustrerons leur fonctionnement. Nous formulerons ensuite le modèle paramétrique décrivant leur fonctionnement, ce qui nous permettra finalement de poser notre problématique.
- La deuxième partie est consacrée à l'état de l'art des solutions existantes et des jeux de données associés. Pour chaque réseau de neurones, nous expliquons les motivations, les choix d'implémentation et les gains apportés. Nous fournissons, avec chaque jeu de données, une explication détaillée du protocole d'acquisition ainsi que plusieurs statistiques. Cela nous permettra d'expliquer nos propre choix quand nous présenterons NEMA.
- La troisième partie détaille notre protocole de tests en commençant par le calcul des étiquettes, des augmentations d'images et des mesures de performances. Puis nous présentons les résultats obtenus par AlexNet, VGG, GoogLeNet, YOLO, PoseNet et BB8 sur LINEMOD, T-LESS et YCB-VIDÉO.
- La quatrième et dernière partie détaille la réalisation de notre jeu de données et les tests réalisés avec ce dernier.

Nous terminons avec une conclusion. Les annexes apportent plus de détails ainsi que des

graphiques présentant les résultats obtenus pour chaque entraînement réalisé. Des réalisations logicielles utilisées pour produire les résultats de cette thèse et utilisées à ITECA sont aussi présentées en annexe.

Informations pour le lecteur

Symboles

Symboles mathématiques ou notations utilisés dans les équations et les schémas ainsi que leur signification.

Notations générales :

- \hat{x} Valeur estimée du (des) paramètre(s) " x ".
- $|X|$ Nombre d'éléments dans l'ensemble " X ".
- X_i " i -ème" sous-élément de X .
- X_Y X défini dans le repère Y .

Repères 2D et 3D :

- \mathbb{I} Repère 2D des images couleur RGB ou RGBA.
- \mathbb{P} Repère 2D des cartes de profondeur.
- \mathbb{B} Repère 2D des images binaires telles que les masques de segmentation.
- \mathbb{M} Repère 3D du modèle 3D d'un objet.
- \mathbb{S} Repère 3D de la scène.
- \mathbb{C} Repère 3D de la caméra.

Paramètres du système optique :

- K Matrice de paramètres intrinsèques de caméra (dimensions : 3×3).
- I Ensemble d'images d'une séquence vidéo ou d'un jeu de données.
- I_i " i -ème" image de la séquence vidéo ou d'un jeu de données.
- S_i Scène vue dans la " i -ème" image".
- M_i Ensemble des objets vus dans la " i -ème" image.
- $M_{i,j}$ " j -ème" objet dans la " i -ème" image.
- $M_{i,j,k}$ " k -ème" point dy " j -ème" objet dans la " i -ème" image.
- $C_{i,j}$ Catégorie du " j -ème" objet dans la " i -ème" image.
- $T_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ Matrice de transformation affine plaçant l'objet $M_{i,j}$ dans le repère \mathbb{C} (dims : 4×4).
- $R_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ Matrice de rotation orientant l'objet $M_{i,j}$ dans le repère \mathbb{C} (dims : 3×3).
- $e_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ Angles d'Euler orientant l'objet $M_{i,j}$ dans le repère \mathbb{C} (dims : 3).
- $q_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ Quaternions orientant l'objet $M_{i,j}$ dans le repère \mathbb{C} (dims : 4).
- $t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ Vecteur de translation positionnant l'objet $M_{i,j}$ dans le repère \mathbb{C} (dims : 3).
- $br_{i,j,\mathbb{I}}$ Rectangle encadrant l'objet $M_{i,j}$ dans la " i -ème" image (dims : 2×2).
- $bb_{i,j,\mathbb{M}}$ Boite encadrante de l'objet $M_{i,j}$ définie dans le repère objet (dims : 8×3).
- $bb_{i,j,\mathbb{C}}$ Boite encadrante de l'objet $M_{i,j}$ projeté dans le repère image (dims : 8×2).
- \hat{I}_i Rendu 3D de la scène S_i vue dans la " i -ème" image.

Paramètres des réseaux de neurones :

- k Dimensions de la fenêtre glissante d'une couche ("*kernel*").
- u Nombre de neurones d'une couche ("*units*").
- s Pas de la fenêtre glissante d'une couche ("*stride*").
- p Largeur des bordures ajoutées à l'entrée d'une couche ("*padding*").
- a Fonction d'activation d'une couche ("*activation*").
- r Proportion de *dropout* à la sortie d'une couche ("*rate*").

Abréviations

Abréviations utilisées et leurs significations.

6DoF	Six degrés de liberté caractérisant la pose d'un objet en 3D.
API	Interface de Programmation des Applications
AR	Réalité Augmentée.
ArUco	Marqueurs de repères de l'Université de Cordoba.
CAO	Conception assistée par Ordinateur.
ChArUco	Marqueurs ArUco disposés en échiquier.
CNN	Réseau de neurones convolutionnel.
HUD	Affichage tête haute (" <i>Head-Up Display</i> " en Anglais).
IoU	Intersection divisé par l'union.
MR	Réalité Mixte.
MR	Réalité Mixte.
NN	Réseau de neurones.
OEM	Fabricant d'équipement d'origine.
VR	Réalité Virtuelle.

Légende des architectures

Code couleur et notations des couches utilisées pour construire les architectures des réseaux de neurones dans les figures de cette thèse.

DONNÉES $W \times H \times D$	Tenseur utilisé pour représenter les données d'entrées ou de sorties du réseau ou d'un sous module.
DENSE $u: 4096, a: \text{ReLU}$	Couche dense complètement connexe avec u unités et une activation a .
CONVOLUTION $k: 3 \times 3, u: 512, s: 1, p: \text{same}, a: \text{ReLU}$	Couche de convolution 2D avec un noyau de k , u filtres, un remplissage p et une activation a .
DÉCONVOLUTION $k: 2 \times 2, u: 512, s: 1, p: \text{same}, a: \text{ReLU}$	Couche de déconvolution 2D avec un noyau de k , un remplissage p et une activation a .
ACTIVATION	Couche d'activation tel que Sigmoid, ReLU, Leaky ReLU, Linéaire ou Softmax.
DROPOUT $r: 50$	Couche de dropout où $r / 1$ des activations sont mises à 0.
NORMALISATION	Couche de normalisation telle que les normalisations par paquets ou normalisations de réponses locales.
ÉCHANTILLONAGE $k: 2 \times 2, s: 2, p: \text{valid}$	Couche de sous-échantillonnage spatiale ou globale telle que Max Pooling, Global Average Pooling et ROI Pooling.
REORGANISATION	Couche de réorganisation telle que les concaténations, additions et autres manipulations de formes sur les tenseurs.
AUTRES	Opérations incluent dans l'architecture qui ne sont pas des couches, par exemple la recherche sélective.

Sommaire

	Page
Résumé	ii
Informations pour le lecteur	vi
Symboles	vi
Abréviations	vii
Légende des architectures	viii
Introduction Générale	2
Introduction	2
Cadre en entreprise	3
Sujet de la thèse	3
Contributions	4
Structure du manuscrit	5
I Contexte, Définitions et Problématiques	6
1 Contexte et Définitions	8
1.1 Détection 3D	8
1.2 Réalité Mixte et Hologrammes	9
1.2.1 Réalité Augmentée	9
1.2.2 Réalité Mixte	10
1.2.3 Réalité virtuelle	10
1.3 Inspirations	10
1.3.1 Fiction	10
1.3.2 Actualité	11
1.4 Matériels et exemples d'applications	13
1.4.1 Tablettes et Smartphones	13
1.4.2 Casques de réalité mixte	14
1.4.3 Conclusion	15

2	Caméra, Images et Rendus 3D pour la Réalité Mixte	16
2.1	Caméra et Images Réelles	17
2.1.1	Caméra et images	18
2.1.2	Objets 3D	20
2.1.3	Positionnement 3D	21
2.1.4	Formation des images	23
2.1.5	Récapitulatif	25
2.1.6	Recadrage	25
2.1.7	Zoom	26
2.2	Rendus 3D	27
2.2.1	Scène 3D	27
2.2.2	Modèles 3D	28
2.2.3	Caméra	29
2.2.4	Positionnement des objets	30
2.2.5	Positionnement de la caméra	32
2.2.6	Projection dans le plan image	32
2.2.7	Formulation Complète	33
3	Problématique	35
3.1	Paramètre Intrinsèques	35
3.2	Paramètre Extrinsèques	36
3.3	Modèle 3D	36
II	État de l'art	38
4	Méthodes de détection	40
4.1	Détection 1D	40
4.1.1	AlexNet	41
4.1.2	VGG-19	43
4.1.3	GoogLeNet	45
4.1.4	Res-Net	45
4.1.5	Xception	48
4.2	Détection 2D	50
4.2.1	Approche Naïve	51
4.2.2	R-CNN	52
4.2.3	Fast R-CNN	54
4.2.4	Faster R-CNN	54
4.2.5	YOLO	55
4.3	Détection 3D	58
4.3.1	Patrons de Pose	58
4.3.2	Forêt de Hough	59
4.3.3	Réseaux estimant directement la pose	59
4.3.4	Correspondances de Points 2D-3D	62
4.3.5	Correspondances de Points 2D-3D Denses	63
4.3.6	Embedding 3D	68

5	Jeux de données	72
5.1	LINEMOD	72
5.1.1	Informations générales	74
5.1.2	Protocole d'acquisition	74
5.1.3	Protocole d'utilisation	76
5.1.4	Intérêt et points forts	76
5.1.5	Limitations	77
5.2	LINEMOD+OCCLUDED	79
5.2.1	Informations générales	79
5.2.2	Protocole d'acquisition	80
5.2.3	Protocole d'utilisation	80
5.2.4	Intérêts et points forts	80
5.2.5	Limitations	82
5.3	T-LESS	83
5.3.1	Informations générales	83
5.3.2	Protocole d'acquisition	84
5.3.3	Protocole d'utilisation	87
5.3.4	Intérêts et points forts	87
5.3.5	Limitations	91
5.4	YCB-VIDEO	92
5.4.1	Informations générales	92
5.4.2	Protocole d'acquisition	93
5.4.3	Protocole d'utilisation	94
5.4.4	Intérêts et points forts	94
5.4.5	Limitations	95
5.5	Autres jeux de données	98
5.5.1	ITODD (MVTec ITODD)	98
5.5.2	Homebrew	99
5.5.3	Falling Things	100
5.5.4	RU-APC (Rutgers APC)	100
5.5.5	IC-BIN	101
5.5.6	IC-MI	102

III Évaluation sur les jeux de données existants 106

6	Données d'apprentissages	108
6.1	Introduction	108
6.2	Images	108
6.2.1	Caméras	108
6.2.2	Images Couleurs	109
6.2.3	Cartes de profondeurs	109
6.3	Étiquettes pour la détection 1D	110
6.3.1	Noms des objets	110
6.3.2	Index des objets	111
6.3.3	Vecteur Logique	111
6.4	Étiquettes pour la détection 2D	112
6.4.1	Rectangle encadrant	113
6.4.2	Masque de segmentation et d'instances	118

6.5	Étiquettes pour la détection 3D	120
6.5.1	Transformations affines, matrices de rotation et vecteurs de translation	120
6.5.2	Angles d'Euler	120
6.5.3	Quaternion	122
6.5.4	Correspondances 2D-3D	122
6.5.5	Correspondances 2D-3D Denses	124
6.5.6	Embedding 2D-3D	125
6.5.7	Conclusion	126
7	Mesures et fonctions de coûts	127
7.1	Mesures et coûts pour la détection 1D	127
7.1.1	Mesures	127
7.1.2	Fonctions de coûts	132
7.2	Mesures et coûts pour la détection 2D	133
7.2.1	Mesures	133
7.2.2	Fonctions de coûts	137
7.3	Mesures et coûts pour la détection 3D	137
7.3.1	Mesures	138
7.3.2	Fonctions de Coûts	141
8	Préparation des images d'entraînements	144
8.1	Recadrage et échelle des images	144
8.2	Substitution des arrière-plans	145
8.3	Ajustements Aléatoires	146
8.3.1	Luminosité	146
8.3.2	Contraste	147
8.3.3	Teinte	147
8.3.4	Saturation	148
8.3.5	Qualité JPEG	149
8.4	Conclusion	149
9	Résultats de détections	150
9.1	Détection 1D	150
9.1.1	VGG-19	151
9.1.2	AlexNet	155
9.1.3	GoogLeNet	156
9.2	Détection 2D	158
9.2.1	YOLO	159
9.3	Détection 3D	161
9.3.1	POSENET	162
9.3.2	BB8	166
10	Analyse	169
IV	NEMA	172
11	Matériel	176
11.1	Caméra	176
11.2	Objets	176

11.3	Plateau tournant	178
11.4	Table et support caméra	179
11.5	Environnement	180
12	Protocole d'acquisition	181
12.1	Calibration de la caméra	181
12.2	Création d'une scène	182
12.3	Création d'une séquence	182
12.4	Acquisition de l'arrière-plan	183
12.5	Acquisition des avant-plans	183
12.6	Comparaison	183
13	Jeu de données	185
13.1	Scène HD	185
13.1.1	Points de vues des objets	185
13.1.2	Position des objets à l'image	186
13.1.3	Visibilité des objets	186
13.1.4	Qualité des étiquettes	187
13.2	Autres Scènes	187
13.3	Comparaison	188
14	Résultats de détection 3D	189
14.1	PoseNet	189
14.2	BB8	191
14.3	Conclusion des expériences	193
	Conclusion	196
15	Rappel du contexte et de la problématique	196
16	Résultats des expériences	198
17	Contributions	201
18	travaux futurs	203
V	Annexes	205
A	Détails des entraînements réalisés	206
A.1	VGG 19 / LINEMOD	207
A.2	VGG 19 / TLESS / #1	208
A.3	VGG 19 / TLESS / #2	209
A.4	VGG 19 / TLESS / #3	210
A.5	VGG 19 / TLESS / #4	211
A.6	VGG 19 / TLESS / #5	212
A.7	VGG 19 / TLESS / #6	213
A.8	VGG 19 / TLESS / #7	214
A.9	VGG 19 / YCB VIDEO / #1	216
A.10	VGG 19 / YCB VIDEO / #2	217

A.11 VGG 19 / YCB VIDEO / #3	218
A.12 ALEXNET / LINEMOD	220
A.13 ALEXNET / TLESS	222
A.14 ALEXNET / YCB VIDEO	224
A.15 GOOGLNET / LINEMOD	226
A.16 GOOGLNET / TLESS	228
A.17 GOOGLNET / YCB VIDEO	230
A.18 YOLO / LINEMOD	232
A.19 YOLO / YCB VIDEO	234
A.20 POSENET / LINEMOD / #1	236
A.21 POSENET / LINEMOD / #2	238
A.22 POSENET / LINEMOD / #3	240
A.23 POSENET / T-LESS	242
A.24 POSENET / YCB-VIDEO	244
A.25 BB8 / LINEMOD / FAIL	246
A.26 BB8 / LINEMOD	248
A.27 BB8 / T-LESS	250
A.28 BB8 / YCB-VIDEO	252
A.29 POSENET / NEMA / #1 (arrière-plans SUN)	254
A.30 POSENET / NEMA / #2 (arrière-plans NEMA)	256
A.31 BB8 / NEMA / #1 (arrière-plans SUN)	258
A.32 BB8 / NEMA / #2 (arrière-plans NEMA)	260

Table des figures

1	WITTY	3
1.1	Exemples d'applications de détection 3D.	8
1.2	Echelle de réalité	9
1.3	Concepts d'applications de réalités augmentées	11
1.4	Matériel actuel de réalité mixte	12
1.5	Applications de réalité mixte sur smartphones ou tablettes	13
1.6	Casques de réalité mixte et applications	14
2.1	Principe de fonctionnement d'une application de réalité mixte	16
2.2	Schéma illustrant l'acquisition d'une image par la caméra.	17
2.3	Comparaison entre une caméra et un œil.	18
2.4	Repère de la caméra et du plan image.	19
2.5	Nuage de points d'un objet (visuel)	20
2.6	Modèle de Phong	20
2.7	Nuage de points d'un objet (numérique)	20
2.8	Illustration du positionnement d'un objet en 3D	22
2.9	Projection du modèle 3D vers image	24
2.10	Illustrations du recrage et de la mise à l'échelle d'une image.	25
2.11	Schéma illustrant la synthèse d'images	27
2.12	Illustration des différents repères 3D utilisés pour la synthèse d'images	28
2.13	Repère de la scène	28
2.14	Repère objet	29
2.15	Repère de la caméra	30
2.16	Schéma illustrant une hiérarchie d'objets	30
3.1	Schéma illustrant le processus de fonctionnement d'une application de réalité mixte	35
3.2	Processus et paramètres considérés du system détection 3D	37
4.1	Schéma illustrant un module de détection 1D	40
4.2	Architecture du réseau de neurones AlexNet	42
4.3	Architecture du réseau de neurones VGG-19	44
4.4	Architecture du réseau de neurones GoogLeNet	46
4.5	Architecture des modules et du réseau ResNet	47
4.6	Architecture d'Xception.	49
4.7	Schéma illustrant un module de détection 2D	50
4.8	Détection 2D naïve	51
4.9	Architectures des réseaux R-CNN, Fast R-CNN et Faster R-CNN.	53
4.10	Architectures des modules de réduction et des réseaux complets YOLO V1 et V2 [94, 92].	56

4.11	Schéma illustrant un module de détection 3D	58
4.12	Architecture du réseau de neurones PoseNet	60
4.13	Architecture de PoseCNN	61
4.14	Architectures de BB8 et Single Shot 6D Pose Prediction.	62
4.15	Architecture complète de DOPE. Les branches 2 à 6 sont optionnelles.	64
4.16	Architecture des modules de DOPE	65
4.17	Architecture de PV-Net.	67
4.18	Utilisation de l'embedding	68
4.19	Architecture de DPOD.	69
4.20	Exemples d'embedding utilisés par CorNet.	70
5.1	Image couleur et carte de profondeur du jeu de données LINEMOD.	73
5.2	Objets du jeu de données LINEMOD	73
5.3	Logiciel utilisé pour l'acquisition de LINEMOD	75
5.4	Positions où apparaissent les objets dans les images de LINEMOD	76
5.5	Points de vus des objets de LINEMOD	77
5.6	Image du jeu de données LINEMOD+OCCLUDED	79
5.7	Objets occultés ou hors champs dans images de LINEMOD+OCCLUDED	81
5.8	Positions où apparaissent les objets dans les images de LINEMOD+OCCLUDED	82
5.9	Points de vus objets de LINEMOD+OCCLUDED	83
5.10	Objets du jeu de données T-LESS	83
5.11	Image du jeu de données T-LESS	84
5.12	Matériel utilisé pour l'acquisition de T-LESS	85
5.13	Positions où apparaissent les objets dans les images de T-LESS	88
5.14	Points de vus des objets de T-LESS	89
5.15	Objets de YCB-OBJECTS et utilisé dans YCV-VIDÉO	92
5.16	Image du jeu de données YCB-VIDÉO	92
5.17	Positions où apparaissent les objets dans les images de YCB-VIDÉO	95
5.18	Points de vu des objets de YCB-VIDÉO	96
5.19	Objets du jeu de données MVTEC ITODD	98
5.20	Matériels utilisé pour l'acquisition de MVTEC ITODD	98
5.21	Objets du jeu de données HOMEBREWED-DB	99
5.22	Séquences du jeu de données HOMEBREWED-DB	99
5.23	Image du jeu de données FALLING THINGS	100
5.24	Rutgers APC	101
5.25	Objets du jeu de données IC-BIN	101
5.26	Objets du jeu de données IC-MI	102
5.27	Exemple d'une image du jeu de données IC-MI	102
6.1	Exemples de caméras de profondeurs.	108
6.2	Exemple d'image couleur et d'une carte de profondeur	109
6.3	Exemples d'étiquettes pour la détection 1D	110
6.4	Illustration de la construction d'un vecteur logique	111
6.5	Calcul du rectangle encadrant à partir d'un modèle 3D	113
6.6	Calcul du rectangle encadrant à partir du rendu	115
6.7	Illustration des étiquettes de YOLO V1 dans une image de T-LESS.	117
6.8	Masque de ségmentation et d'instances	119
6.9	Illustration de la rotation avec une matrice de rotation ou un angle d'euler	121
6.10	Exemple de la boîte encadrante du Canard de LINEMOD.	123
6.11	Illustration des étiquettes de PVNet	125

6.12	Illustration de l'embedding cylindrique et sphérique	125
7.1	Illustration des différents cas de figures entre prédictions et étiquettes.	127
7.2	Illustration du calcul de l'intersection sur l'union	133
7.3	Variation de l'IoU par rapport au décalage entre deux lignes ou rectangles	135
7.4	Illustration de plusieurs fonctions de coûts.	136
8.1	Illustration de la substitution de l'arrière-plan d'une image	145
8.2	Illustration de l'ajustement de la luminosité sur une image.	146
8.3	Illustration de l'ajustement du contraste sur une image	147
8.4	Illustration de l'ajustement de la teinte sur une image	148
8.5	Illustration de l'ajustement de la saturation sur une image	148
8.6	Illustration de l'application de la compression JPEG sur une image	149
9.1	Schéma illustrant un module de détection 1D (rappel)	150
9.2	Schéma illustrant un module de détection 2D (rappel)	158
9.3	Schéma illustrant un module de détection 3D (rappel)	161
9.4	Entraînement 1-7 de PoseNet sur T-LESS	163
9.5	Entraînement 9-14 de PoseNet sur T-LESS	164
9.6	BB8 avec des coordonnées en pixels sur LINEMOD	167
10.1	Exemples d'images où des occlusions induisent GoogLeNet en erreur	169
10.2	Illustration du vide sémantique de T-LESS	170
10.3	Exemples d'images où GoogLeNet confond les deux pinces de YCB-VIDÉO.	171
10.4	Image de notre jeu de données NEMA	174
11.1	Caméra choisie pour photographier notre jeu de données	177
11.2	Moteur NEMA utilisé dans notre jeu de données	177
11.3	Dimensions standards d'un moteur NEMA 17	178
11.4	Plateau tournant conçu et utilisé	179
11.5	Table d'acquisition conçue et utilisée	180
12.1	Schématisation du protocole d'acquisition.	181
13.1	Captured points of views of NEMA parts.	185
13.2	Localisation des objets dans les images de NEMA	186
13.3	Présentation des autres scènes de NEMA	188
14.1	Résultats de PoseNet sur NEMA	190
14.2	Résultats de BB8 sur NEMA	192

Liste des tableaux

4.1	Résultats de classification d'images	50
4.2	Résultats de détection d'objets	57
4.3	Résultats d'estimation de la pose d'objets sur LINEMOD	70
5.1	Information générale sur le jeu de donnée LINEMOD.	73
5.2	Visibilités des objets du jeu de données LINEMOD	78
5.3	Erreurs de profondeurs du jeu de données LINEMOD	78
5.4	Visibilités des objets du jeu de données LINEMOD+OCCLUDED	81
5.5	Erreurs de profondeur du jeu de données LINEMOD+OCCLUDED	81
5.6	Résolution des images de T-LESS.	86
5.7	Visibilités des objets du jeu de données T-LESS (test)	90
5.8	Statistiques d'échelles et d'aires des objets dans les images de T-LESS	90
5.9	Erreurs de profondeur du jeu de données T-LESS (entraînement)	90
5.10	Erreurs de profondeur du jeu de données T-LESS (test)	90
5.11	Visibilités des objets du jeu de données YCB-VIDÉO (entraînement)	94
5.12	Visibilité des objets du jeu de données YCB-VIDÉO (test)	94
5.13	Erreurs de profondeur du jeu de données YCB-VIDÉO (entraînement)	96
5.14	Erreurs de profondeur du jeu de données YCB-VIDÉO (test)	96
5.15	Nombre de photographies de chaque objet du jeu de données IC-MI	102
9.1	Résultats de VGG-19 sur T-LESS selon la préparation des images	153
9.2	Résultats de VGG-19 sur YCB VIDÉO selon la préparation des images	154
9.3	Pourcentage d'images de YCB-VIDÉO correctement classifiées par VGG.	154
9.4	Résultats de classifications d'images	158
9.5	Résultats de PoseNet.	166
9.6	Résultats de BB8.	168
12.1	Paramètres intrinsèques de la caméra Intel Realsense d453i.	182
13.1	Visibilité des pièces du jeu de données NEMA	186
13.2	Erreur de recalage des pièces de NEMA	187
14.1	Résultats complets de PoseNet.	193
14.2	Résultats complets de BB8.	193

Introduction Générale

Introduction

L'industrie a déjà connu trois révolutions majeures. Lors de la première, le charbon et la vapeur ont permis de mécaniser la production. La seconde révolution était étroitement liée au pétrole et à l'automobile. On a vu apparaître à ce moment les premières chaînes de production en série. La dernière révolution électrique, informatique et robotique a démultiplié les capacités de production.

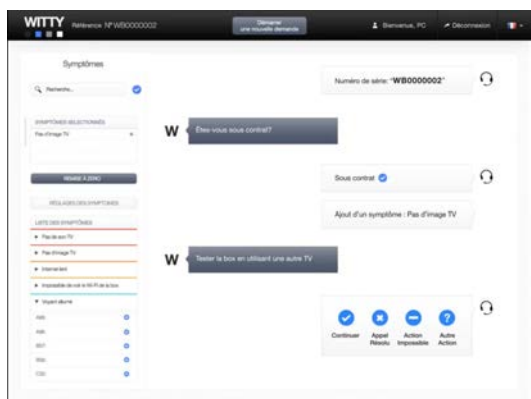
Ces trois révolutions ont mis à disposition des entreprises des moyens leurs permettant de produire massivement et rapidement. Mais ce gain de production a aussi des contraintes que les entreprises doivent adresser. Les machines sont majoritairement des pièces uniques ou issues de petites séries. Elles comportent des pièces d'usure et peuvent tomber en panne. Des guides et des aides à la maintenance sont fournis par les fabricants d'équipements originaux (OEM) qui les conçoivent souvent sous forme papier allant de quelques pages à plusieurs classeurs. Les plus innovants fournissent ces guides au format PDF ou en ligne.

Lorsqu'une machine tombe en panne ou nécessite une maintenance périodique, la production est arrêtée pour permettre son entretien. Tout arrêt de la production a un coût pour l'entreprise. L'opérateur chargé de la réparation doit diagnostiquer la panne et conduire les réparations. La documentation peut l'aider dans la réalisation de son diagnostic et lui détailler les actions à mener. Mais, pour y arriver, il doit trouver l'information et la comprendre. Ce qui est facilité par des schémas et des dessins techniques. Puis, il doit réaliser les réparations en respectant des contraintes de sécurité, d'hygiène et de traitement des déchets. Tout erreur de diagnostic, de compréhension ou d'exécution représente une perte de temps potentiel, pire, un risque d'endommager plus encore la machine, ou un drame, si l'opérateur se blesse. Dans l'agro-alimentaire, chaque intervention sur la production doit faire l'objet d'un rapport détaillé à présenter à l'administration. Déjà certains outils numériques ont permis de dématérialiser les guides de production et proposent des moteurs de recherche intelligents qui amènent les opérateurs vers la meilleure solution. Les vues 3D animées permettent de mieux illustrer les actions de démontage et de remontage. Ces outils peuvent aussi rédiger en partie les rapports au format numérique, même s'il faut toujours les imprimer pour les transmettre à l'administration.

Aujourd'hui, la mode est à la customisation, que ce soit pour les voitures (couleurs, finitions, motorisations, équipements), les chaussures (couleurs, matériaux, messages personnalisés) ou tout autre produit. Des normes écologiques sont imposées pour répondre à l'urgence climatique en même temps que le coût de l'énergie augmente. C'est pour répondre à ces défis qu'une quatrième révolution est en train d'avoir lieu. Les données communiquées par les robots sont traitées et analysées pour optimiser les coûts et les consommations. Des intelligences artificielles permettent de prédire les pannes, d'anticiper les opérations de maintenance et d'optimiser la production pour en réduire les coûts. Les outils de management sont aussi connectés à la production pour que l'information circule à travers toutes les couches de l'entreprise.

C'est la révolution 4.0.

Cadre en entreprise



(1) WITTY



(2) WITTY 3D

FIGURE 1 – Logiciel d'aide à la maintenance WITTY.

ITECA développe des outils logiciels adaptés aux usines 4.0 et elle accompagne ses clients dans la transformation vers l'usine du futur. En particulier *WITTY 3D* (qui signifie "drôle, spirituel, fin" en anglais) se concentre sur la résolution des opérations de maintenance et la prévision des pannes. A son cœur, se trouve le moteur d'IA décisionnel développé par SpirOps, dont ITECA a la licence exclusive pour les applications industrielles 4.0. C'est le même que celui utilisé dans le prototype de véhicule autonome de Peugeot. Il est couplé avec une scène 3D qui permet d'identifier visuellement l'origine de la panne et de guider le technicien dans ses réparations. Ce qui permet de résoudre plus rapidement le problème. Ce logiciel fonctionne déjà sur PC, tablettes et smartphones. En 2015 la sortie des HoloLens produites par Microsoft a motivé ITECA à adapter *WITTY 3D* sur ce nouveau support matériel. Les bénéfices pour les utilisateurs seraient nombreux. Premièrement, le contrôle par les gestes et la voix ainsi que l'affichage tête haute permettent de libérer les mains. Pour un technicien en intervention, c'est un avantage majeur car il peut travailler sur la machine réelle tout en interagissant avec l'application. Deuxièmement, la visualisation 3D peut être incrustée dans l'environnement réel. Ainsi la panne est localisée sur la machine réelle et les instructions de réparation sont contextualisées. Le technicien n'a plus besoin de relocaliser l'information qui lui était présentée, ce qui réduit sa charge cognitive et limite les erreurs d'interprétation. C'est dans le cadre de ce projet que ITECA s'est rapprochée du LaBRI, et qu'ensemble, ils ont proposé ce sujet de thèse en Convention Industrielle de Formation par la REcherche (CIFRE).

Sujet de la thèse

Cette thèse a pour objectif d'identifier les verrous technologiques encore à résoudre lors du développement d'une application d'aide à la maintenance sur casque de réalité mixte et de les adresser. Pour que les objets virtuels et les informations affichées soient incrustés sur les objets réels il faut que l'application soit capable de reconnaître ces objets et d'estimer leurs positions et orientations en 3D par rapport au référentiel 3D du casque. L'application dispose des images des caméras et des données des capteurs intégrés au casque. Il faut donc mettre en œuvre un système de reconnaissance d'objets et d'estimation de leurs six degrés de liberté. Le système doit pouvoir fonctionner sur le casque en plus de l'application et de l'IA décisionnelle. Or, ces casques disposent d'une puissance de calcul et d'une mémoire limitées.

Contributions

Au cours de la période de thèse nous avons publié deux articles :

Limitations of Metric Loss for the Estimation of Joint Translation and Rotation

Philippe Pérez de San Roman, Pascal Desbarats, Jean-Philippe Domenger, Axel Buendia
International Joint Conférence on Computer Vision, Imaging and Computer Graphics Theory
and Applications (VISAPP)
25-27 Février 2019
Prague, République Tchèque
<https://www.scitepress.org/Link.aspx?doi=10.5220/0007525005900597>

Nous avons soumis cet article après avoir réalisé nos tout premiers tests d'implémentation de PoseNet. Dans ces tests, nous utilisons les images d'entraînements de T-LESS pour entraîner et valider notre implémentation de PoseNet. Nous n'avons pas encore implémenté d'augmentations d'images, ni de soustractions des arrières-plans qui sont noirs dans ces images. Ces tests ont fait ressortir un comportement problématique de la fonction de coût utilisée pour entraîner PoseNet qui comporte deux termes. Le premier terme mesure l'erreur d'estimation de la rotation 3D comme une différence entre deux quaternions : étiquette versus prédit. Le deuxième fait de même pour la translation 3D qui est exprimée comme un vecteur en mètres. Les deux termes sont équilibrés manuellement par deux hyper-paramètres. Nos expériences ont montré que, dans le cas où les images d'entraînement ne sont pas assez variées, la translation domine la fonction de coût et la rotation n'est pas apprise. En jouant sur la pondération pour favoriser la rotation, celle-ci peut être apprise par le réseau dans les premiers pas de l'entraînement, mais elle est oubliée au profit de la translation dans la suite de l'entraînement.

Les expériences que nous avons réalisées par la suite dans cette thèse montrent que PoseNet est très bon pour prédire une translation précise, mais a plus de difficultés à prédire une rotation précise. Cela peut s'expliquer par le conflit entre les deux termes, sur lequel la translation l'emporte, que nous avons présenté dans cet article.

NEMA : 6-DoF Pose Estimation Dataset for Deep Learning

Philippe Pérez de San Roman, Pascal Desbarats, Jean-Philippe Domenger, Axel Buendia
International Joint Conférence on Computer Vision, Imaging and Computer Graphics Theory
and Applications (VISAPP).
6-8 Février 2022
En ligne
<https://www.scitepress.org/Link.aspx?doi=10.5220/0010913200003124>

Cette publication regroupe notre analyse des jeux de données de l'état de l'art et présente notre jeu de données NEMA. Pour notre jeu de données nous détaillons le matériel utilisé. Cela inclut le matériel que nous avons conçu comme le plateau tournant, la table et les objets photographiés. Notre jeu de données est le premier à proposer des modèles aux formats paramétriques et surfaciques. Nous illustrons ensuite les images obtenues et détaillons les mêmes statiques que celles calculées sur les jeux de données de l'état de l'art. Nous montrons ainsi que notre jeu de données propose plus d'images avec une meilleure résolution, à des points de vue plus denses. L'erreur mesurée sur NEMA prouve aussi qu'il propose des étiquettes plus précises que ses prédécesseurs. De plus il inclut les arrières-plans qui permettent d'effacer les marqueurs ce qui est inédit.

Les expériences que nous avons réalisées par la suite dans cette thèse montrent que NEMA permet d'entraîner des réseaux de neurones à des précisions jamais atteintes, et que les arrières-plans jouent un rôle significatif dans ce gain de précision.

Structure du manuscrit

La suite de cette thèse est décomposée en quatre parties :

- La première partie détaille le sujet de la thèse et la problématique. Nous apporterons un certain nombre de définitions et expliquerons nos inspirations ainsi que les retours d'expériences de nos partenaires industriels. Nous y présenterons les supports matériels et illustrerons leur fonctionnement. Nous formulerons ensuite le modèle paramétrique décrivant leur fonctionnement, ce qui nous permettra finalement de poser notre problématique.
- La deuxième partie est consacrée à l'état de l'art des solutions existantes et des jeux de données associés. Pour chaque réseau de neurones, nous expliquons les motivations, les choix d'implémentations et les gains apportés. Nous fournissons avec chaque jeu de données une explication détaillée du protocole d'acquisition ainsi que plusieurs statistiques. Cela nous permettra d'expliquer nos propre choix quand nous présenterons NEMA.
- La troisième partie détaille notre protocole de tests en commençant par le calcul des étiquettes, des augmentations d'images et des mesures de performances. Puis nous présentons les résultats obtenus par AlexNet, VGG, GoogLeNet, YOLO, PoseNet et BB8 sur LINEMOD, T-LESS et YCB-VIDÉO.
- La quatrième et dernière partie détaille la réalisation de notre jeu de données et les tests réalisés avec ce dernier.

Nous terminons avec une conclusion qui rappellera les résultats marquants produits durant cette thèse et proposera des perspectives.

Les annexes apporteront plus de détails et de graphiques présentant les résultats calculés pour chaque entraînement réalisé. Les réalisations logicielles utilisées pour produire les résultats présentés dans la thèse et utilisés à ITECA sont aussi présentées en annexe. Nous avons aussi présenté des logiciels dont nous avons participé au développement et qui préparent le port de *WITTY 3D* sur casque de réalité mixte.

Première partie

Contexte, Définitions et Problématiques

Introduction

Cette première partie d'introduction a pour but d'apporter au lecteur toutes les informations nécessaires pour aborder cette thèse.

Dans un premier chapitre, nous présenterons les définitions des notions au cœur de cette thèse comme la détection 3D ou la réalité augmentée. Ce chapitre inclut aussi les inspirations qui traduisent, dans la culture populaire, ces notions et qui illustrent aussi, pourquoi nous nous y intéressons. Et enfin, ce chapitre présente les matériels à notre disposition pour mettre aujourd'hui en œuvre des applications de réalité mixte.

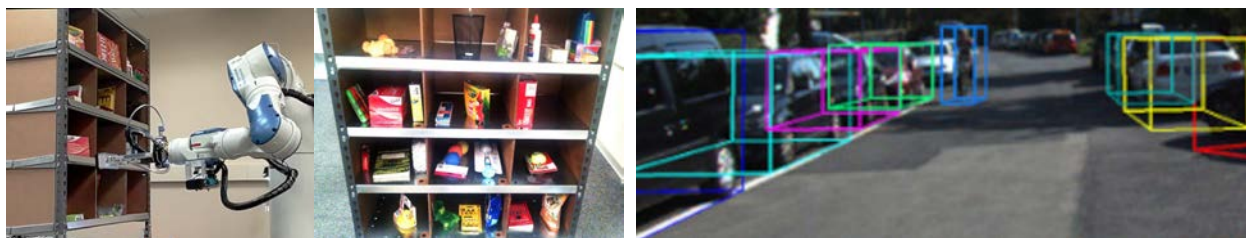
Le deuxième chapitre est consacré au fonctionnement théorique des casques de réalité mixte et à l'utilisation que nous voulons en faire. Premièrement, nous voulons utiliser la caméra du casque pour reconnaître et estimer la position et l'orientation d'objets (estimation des 6DoF). Deuxièmement, nous souhaitons afficher des visuels superposés à, ou placés proche de l'objet réel. Pour que le processus complet fonctionne, il nous faut bien comprendre le fonctionnement de chacun des deux points ci-dessus. Le premier point est lié à la projection perspective qui a lieu quand une caméra capture une image. L'estimation des six degrés de liberté va chercher les paramètres latents du système d'équations qui modélise le fonctionnement de la caméra. Le deuxième point est lui exactement l'inverse, à savoir, comment recréer une image à partir d'une scène 3D. Ce problème est déjà résolu puisqu'il s'agit de la génération de rendus 3D. Nous pouvons donc utiliser n'importe quelle bibliothèque logicielle 3D (OpenGL, Direct 3D, Vulkan) et bénéficions des accélérations matérielles des GPU et APU.

Enfin, le troisième chapitre détaille la problématique : l'estimation des six degrés de liberté.

Chapitre 1

Contexte et Définitions

1.1 Détection 3D



(1) Un robot qui prend en main des objets[96]. (2) Détection 3D de voitures pour la conduite autonome[42].

FIGURE 1.1 – Exemples d'applications de détection 3D.

La détection 3D a pour but de fournir à une application la capacité de comprendre son environnement. Contrairement à la seule détection, qui se contente d'identifier les objets, c'est à dire de savoir lesquels sont présents ou non, la détection 3D cherche aussi à les localiser en 3D et permet donc à l'application de modéliser l'environnement dans lequel elle évolue. Elle ne localise pas les objets dans le plan image mais bien en 3 dimensions dans l'environnement réel. Pour y parvenir, l'application peut utiliser plusieurs capteurs comme des caméras ou des capteurs de distances.

Par exemple, un robot qui se déplace a besoin de détecter les limites de son environnement, les obstacles, de les identifier et de les localiser par rapport à lui-même. Les voitures autonomes, qui sont des robots, utilisent plusieurs caméras, des scanners lasers LiDARs et des capteurs de distances. La figure 1.1 présente des exemples supplémentaires.

Une autre mise en œuvre possible avec la détection 3D est la réalité mixte. L'application connaît son environnement et peut donc afficher directement des informations et des objets virtuels au milieu de celui-ci. Ces objets virtuels peuvent notamment être interactifs.

La réalité mixte et ses applications suscitent beaucoup d'intérêt dans l'industrie. L'informatique y est déjà largement utilisée depuis la modélisation des produits, jusqu'au suivi de leur vie, en passant par leurs productions. La réalité mixte permettra par exemple aux ingénieurs de concevoir des produits tout en pouvant les visualiser à l'échelle 1:1 et même de les manipuler, de les essayer. Ils vont pouvoir valider leurs conceptions ou les modifier directement dans un même outil. Les boucles itératives de conception seront donc plus courtes et plus intuitives. Ils vont donc gagner en performance. Pour la maintenance, la réalité mixte permet d'accompagner l'opérateur en l'aidant à visualiser la cause d'une panne, même sur

les pièces internes d'une machine. Des informations de contexte peuvent aussi être affichées à proximité des objets sur lesquels l'utilisateur travaille.

1.2 Réalité Mixte et Hologrammes

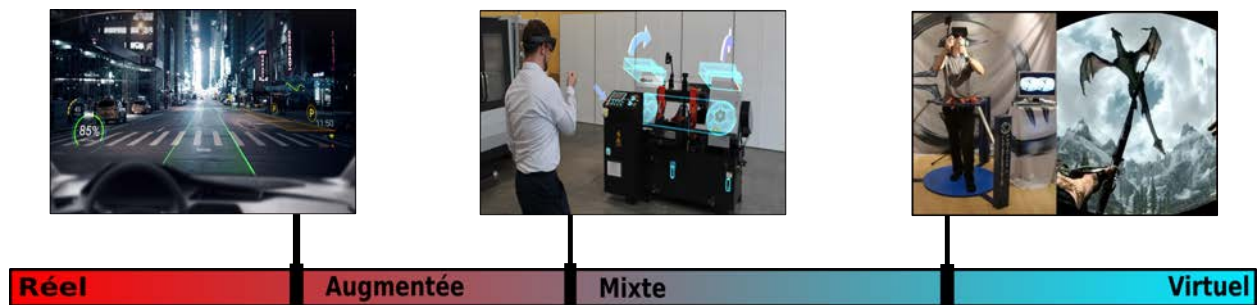


FIGURE 1.2 – Echelle de réalité permettant de positionner et comprendre les différences entre réalité, réalité augmentée, réalité mixte, et réalité virtuelle [79]. La réalité virtuelle pure exclut tout élément réel, alors que réalité mixte combine réel et virtuel de façon si naturelle que l'utilisateur peut ne pas discerner la différence entre les deux. La réalité augmentée est différente de la réalité mixte car les objets virtuels se superposent aux réels et il n'est pas possible d'interagir de façon naturelle avec eux.

Les applications de réalité mixte, aussi appelées applications de réalité hybride, désignent les applications dans lesquelles des objets virtuels sont affichés en trois dimensions et se fondent dans la perception qu'ont les utilisateurs de l'environnement réel. De plus, elles permettent aux utilisateurs d'interagir et de manipuler ces objets virtuels en temps réel et de les faire interagir avec des objets réels. Les objets virtuels dans les applications de réalité mixte sont appelés hologrammes par analogie avec les techniques optiques pour les produits. A la différence des applications de réalité virtuelle, les utilisateurs ne sont pas coupés du monde réel. Et contrairement à la réalité augmentée, les objets virtuels affichés sont incrustés à l'environnement réel et sont interactifs.

La figure 1.2 présente les différences entre réalité augmentée, mixte et virtuelle en les positionnant les unes par rapport aux autres sur une échelle allant du réel au virtuel. A gauche se trouve le réel. Plus on avance vers la droite, plus d'éléments virtuels sont présents dans l'environnement de l'utilisateur et plus ils sont incorporés au réel. Tout à droite on arrive à la réalité virtuelle où le réel n'est plus perceptible et il n'y a que du contenu virtuel.

1.2.1 Réalité Augmentée

La réalité augmentée affiche des informations sur des supports physiques placés dans le champ de vision de l'utilisateur. Il n'est possible d'interagir avec ces affichages autrement que par le biais de menus ou grâce à des commandes numériques. L'industrie déploie déjà cette technologie. Par exemple, les voitures récentes affichent des informations de conduite sur une vitre placée dans le champ de vision du conducteur, ce qui était déjà mis en œuvre dans l'aéronautique depuis un certain temps. On parle d'affichage tête haute ("head up display" ou HUD en anglais).

1.2.2 Réalité Mixte

La réalité mixte ne cherche pas à isoler l'utilisateur dans le virtuel, elle amène le virtuel à l'utilisateur. Comme la réalité augmentée, les objets virtuels se superposent à la perception de l'environnement réel mais sont détachés de tout support physique d'affichage ce qui permet de les positionner dans l'environnement réel à l'échelle 1:1 et à l'utilisateur d'interagir avec eux de manière naturelle. Il est possible de manipuler les objets virtuels, de les déplacer, etc, comme on le fait avec des objets réels. Des applications industrielles de la réalité mixte commencent à apparaître, comme les outils d'aide à la maintenance Microsoft Dynamics 365 Guides [78].

1.2.3 Réalité virtuelle

La réalité virtuelle pour finir est à l'opposé du réel. Elle immerge l'utilisateur dans un environnement complètement virtuel qui n'intègre aucun élément réel. Pour ce faire, l'utilisateur est isolé du monde réel, et ses sens sont stimulés artificiellement pour lui donner la perception du monde virtuel. Par exemple des casques de réalité virtuelle comme le HTC Vive où l'Oculus se substituent à la vue et à l'ouïe du porteur grâce à des écrans placés devant les yeux et un casque anti-bruit sur ses oreilles. Plus les sens de l'utilisateur sont couverts, plus l'immersion est totale. Elle est principalement utilisée dans le domaine vidéo-ludique qui l'a démocratisé, et commence à trouver des applications industrielles pour la formation et la visualisation 3D.

1.3 Inspirations

1.3.1 Fiction

Les écrivains et les réalisateurs de films et de séries télévisées ont imaginé un grand nombre d'appareils et d'applications de réalités mixtes qui constituent de très bons exemples, dont les applications modernes s'inspirent.

Dans beaucoup de films de science fiction, comme Star Wars, des projecteurs holographiques peuvent dessiner des objets volumiques et même tangibles en manipulant la lumière. Ces projections constituent un nouveau moyen de communication. Dans Blade Runner 2048 l'hologramme est poussé encore plus. Il donne corps à une IA qui est un personnage à part entière. Ce type d'hologramme par projection, au sens traditionnel du terme, est à la base de la définition de la réalité mixte : du virtuel est ajouté au réel en trois dimensions, et l'utilisateur peut interagir naturellement avec.

Des implants pour le cerveau ou les yeux ont aussi été imaginés. Ils pourraient manipuler notre vue, notre ouïe et même tous nos sens. On parle alors d'humain augmenté, capable de percevoir plus d'informations, et même capable de se projeter dans un monde virtuel. C'est par exemple le cas dans la série Altered Carbon. Dans des jeux vidéo comme Cyberpunk ou Deus Ex, des implants sont des modifications corporelles cybernétiques et permettent d'augmenter les personnages et leurs donner accès à des outils de réalité augmentée, mixte et virtuelle. Néanmoins, seule la ou les personnes équipées de ces dispositifs peuvent percevoir les informations et les objets virtuels, et ils ne sont pas affichés dans le monde réel, mais seulement dans la perception que les utilisateurs en ont. Cette technique tend donc plus de la réalité virtuelle que de la réalité mixte.

De la matière programmable, donc capable de prendre n'importe quelle forme et de remplir n'importe quelle fonction a aussi été imaginée dans des films comme Man Of Steel



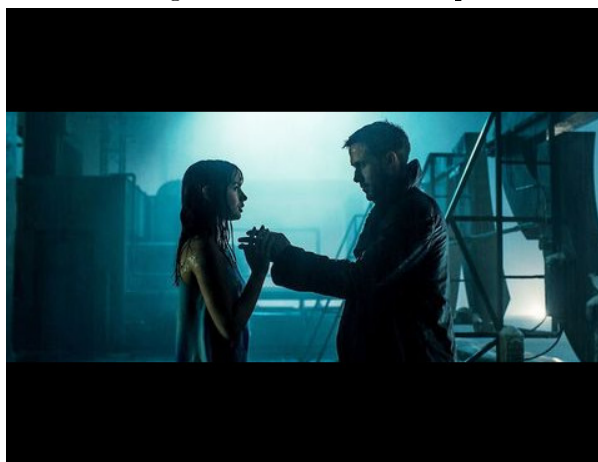
(1) Star Wars IV, Un Nouvel Espoir : R2D2 projette un hologramme enregistré par la princesse Leia à Obi Wan Kenobi et Luke Skywalker.



(2) Altered Carbon, Saison 1, Épisode 1 : Un implant oculaire permet à Takeshi Kovacs de voir des hologrammes faisant de la promotion.



(3) Man of Steel : De la matière programmable sert de console permettant de visualiser et d'interagir avec l'utilisateur.



(4) Blade Runner 2049 : L'officier K et sa compagne nommée Joi : une intelligence artificielle holographique.

FIGURE 1.3 – Exemples et concepts d'applications de réalités augmentées développés dans des films et des séries TV. La réalité mixte peut servir à développer de nouvelles applications de communication, de nouvelles interfaces utilisateur et même donner corps à des intelligences artificielles.

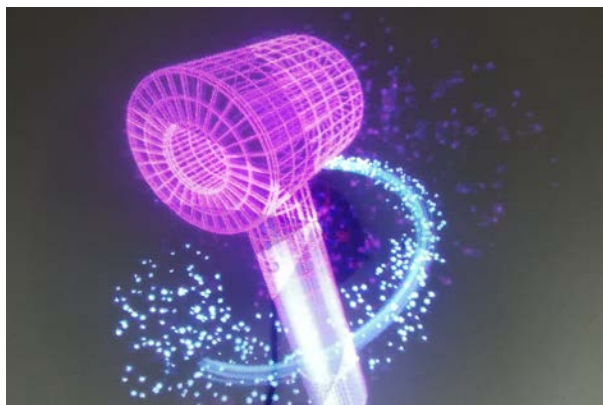
ou des séries comme Star Trek. Cette dernière tend fortement au réel puisqu'on parle bien de matière avant tout. Du fait qu'elle soit programmable, elle peut servir de support d'affichage pour du virtuel et permet donc aussi de mettre en œuvre des applications de réalité mixte.

La figure 1.3 illustre certains de ces concepts.

1.3.2 Actualité

Ces œuvres d'anticipation sont aussi une source d'inspiration pour les chercheurs et les entreprises qui travaillent à faire de ces concepts des réalités.

Des implants cochléaires permettent déjà de stimuler l'ouïe de patients atteints de surdité. On peut donc espérer que le même type d'implant existera pour rendre la vue au patient aveugle. Les yeux sont des organes beaucoup plus compliqués que nos oreilles. En effet, ils traitent des informations beaucoup plus complexes et produisent des signaux nerveux tout aussi complexes. Reproduire artificiellement ces signaux nerveux est donc d'autant plus



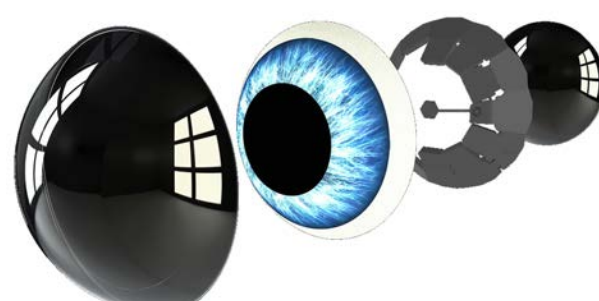
(1) Hologramme décoratif créé par des diodes électroluminescentes monté sur un rotor. L'image se forme grâce à la persistance rétinienne [40].



(2) Hologramme sur eau aux fêtes des lumières 2018 à Lyon réalisés par Aerosculpture et Euro-sono [3, 32].



(3) Hologramme en trompe l'œil affiché sur les 4 côtés d'un trapèze de plexiglass donnant une impression de volume [119].



(4) Lentille de réalité augmentée MOJO Lens [122] capable d'afficher de petites images directement dans la lentille.

FIGURE 1.4 – Innovation technologique d'actualités permettant de mettre en œuvre des applications de réalité augmentée.

compliqué. De plus, les implants cochléaires ne stimulent pas le nerf auditif, ils stimulent la cochlée qui est l'organe qui traduit les vibrations de l'air en signaux nerveux. Stimuler directement un nerf avec les techniques modernes peut finir par l'abîmer. Il n'existe aujourd'hui aucune interface avec la rétine ce qui explique qu'aucun implant permettant de restaurer la vue n'existe encore.

A défaut de pouvoir rendre la vue, plusieurs entreprises travaillent sur des prototypes de lentilles connectées pour l'augmenter. C'est ce que propose MOJO Lens [122] dont les lentilles peuvent afficher des images directement en superposition à notre champ de vision. Les principaux challenges à relever pour de telles lentilles sont les mêmes que dans nos appareils mobiles. À savoir, comment embarquer dans de si petits objets le matériel nécessaire à leur fonctionnement : un processeur capable de produire les images, une antenne pour communiquer avec d'autres appareils, et une batterie pour alimenter le tout. De plus, ces lentilles se contentent d'afficher des images, elles ne sont pas capables de proposer des interactions du fait de l'absence de caméra ou d'autres capteurs. D'où l'importance de l'antenne qui doit permettre de déporter et sous-traiter la collecte d'informations et la commande des lentilles. Ce sont purement des dispositifs de réalité augmentée.

Pour ce qui est des projecteurs holographiques, ils en existent mais pour créer des images

en trois dimensions ils font soit appel à des prismes ou des miroirs optiques, ou alors ils projettent leurs images sur de l'eau (vapeur ou gouttes) ou de la fumée. La technique reposant sur des prismes produit une image à l'intérieur d'une boîte trapézoïdale fermée. Elle a notamment été déployée dans les restaurants McDonald's pour présenter les "Happy Meal" aux enfants à l'entrée de certains restaurants [41]. On peut aussi acheter à bas coût des prises en plastique se posant sur un téléphone portable pour produire de petits hologrammes. Du fait que ces hologrammes sont produits dans une boîte, on ne peut pas interagir avec eux, et ils ne peuvent pas se fondre dans l'environnement réel. Ces dispositifs sont donc des écrans en 3 dimensions. Ceux projetés sur de l'eau et de la fumée ne peuvent être projetés que sur de l'eau ou de la fumée. Ils ne peuvent donc être mis en œuvre que sur des fontaines où dans des lieux prévus à cet effet. De plus, il faut que la luminosité ambiante soit faible et ils ont l'aspect du support sur lequel ils sont projetés, voluptueux pour la fumée et granuleux pour l'eau. Ils sont donc principalement utilisés dans le domaine de l'évènementiel. A noter qu'il existe aussi des hologrammes utilisant la persistance rétinienne grâce à des diodes électro-luminescentes placées sur un rotor. Cette technique permet de dessiner des images lumineuses, utilisées pour créer des horloges ou des affichages décoratifs (en plus de brasser de l'air et de faire du bruit). Encore une fois, l'hologramme affiché n'est qu'un trompe l'œil complètement dépendant de son support d'affichage. Cette technique produit donc de la réalité augmentée.

Plusieurs de ces innovations sont présentées en figure 1.4.

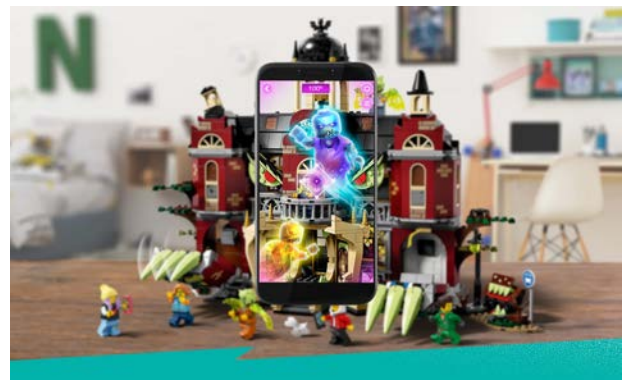
1.4 Matériels et exemples d'applications

Des applications de réalités mixtes existent et sont distribuées. On distingue deux techniques principalement utilisées pour développer ces applications : 1. Les tablettes et smartphones. 2. Les casques de réalité mixte.

1.4.1 Tablettes et Smartphones



(1) Application IKEA Place [59] permettant de visualiser le positionnement d'un meuble dans son intérieur.



(2) Jeux de construction LEGO Hidden-Side [70] ou des animations sont jouées sur le téléphone ou la tablette.

FIGURE 1.5 – Exemples d'applications de réalité mixte sur smartphones ou tablettes.

La première catégorie d'application utilise une tablette ou un smartphone équipé d'une caméra. La vidéo de la caméra est affichée sur l'écran de façon à ce que l'utilisateur puisse voir ce qui se trouve devant la tablette. Les objets virtuels sont dessinés par-dessus l'image

affichée. La centrale inertielle est utilisée pour localiser la tablette par rapport à son environnement et la suivre au cours des déplacements. On ajoute souvent des marqueurs, des mires de calibration ou des objets connus dans l'environnement de travail pour aider la tablette à se positionner. Sinon l'application peut aussi détecter les grandes surfaces planes, comme les sols et les murs, où l'on pourra placer les hologrammes. Mieux la tablette se positionne dans l'environnement, meilleure sa capacité de produire des images se superposent ou s'incrudent dans la vidéo de l'environnement, et donc plus convaincante sera la réalité mixte produite.

Par exemple, l'application mobile IKEA Place [59] permet de visualiser sur téléphone ou tablette les meubles IKEA dans votre intérieur ce qui permet à l'utilisateur de se rendre compte de la place que prendra un meuble. C'est un excellent outil commercial qui permet de toucher une nouvelle clientèle en la faisant jouer avec des meubles IKEA. LEGO a proposé un thème exclusif et limité lors de Halloween 2020 : Hidden Side [70]. Ces ensembles LEGO proposent aux joueurs de scanner avec leurs téléphones ou tablettes les constructions LEGO qui affichent alors des animations. Ces deux exemples d'application sont présentés en figure 1.5.

1.4.2 Casques de réalité mixte



(1) Daqri.



(2) Microsoft HoloLens 2.



(3) ODG.



(4) Fragments : jeux d'enquête en réalité mixte pour hololens.



(5) Microsoft Dynamics 365 Guides : Application d'aide à la maintenance pour hololens 2 [78].

FIGURE 1.6 – Exemples de casques de réalité mixte et applications.

La seconde catégorie utilise un appareil dédié, un casque ou une paire de lunettes, disposant d'écrans transparents qui sont placés devant les yeux de l'utilisateur le portant. Les écrans étant transparents, l'utilisateur peut continuer de voir son environnement et les hologrammes sont affichés sur les écrans et se superposent à ce qu'il voit. De la même façon que pour la technique précédente le casque dispose d'une ou plusieurs caméras lui permettant de se localiser. Ils peut aussi intégrer toutes sortes de capteurs comme une Kinect pour reconstruire un maillage de l'environnement dans lequel il se trouve, ou encore un oculomètre capable de suivre le mouvement des yeux ("eye-tracker" en anglais) pour permettre aux hologrammes de réagir si l'utilisateur les regarde. Ils sont aussi fournis avec des outils logiciels

qui grâce à ces différents capteurs permettent aussi de reconnaître les gestes de l'utilisateur mais aussi les surfaces comme les murs, tables ou chaises qui peuvent servir d'emplacement pour des hologrammes.

La figure 1.6 présente plusieurs modèles de casques et applications utilisant ces appareils.

1.4.3 Conclusion

Ces deux techniques sont les seules qui permettent aujourd'hui de déployer à grande échelle des applications de réalité mixte. Elles permettent toutes les deux d'afficher de façon robuste des informations en 3 dimensions, qui peuvent même être intégrées sur des surfaces type ou des objets connus. Dans la partie suivante nous présentons le fonctionnement de ces deux techniques, et mettons en évidence les points qui restent à perfectionner.

Chapitre 2

Caméra, Images et Rendus 3D pour la Réalité Mixte

Introduction

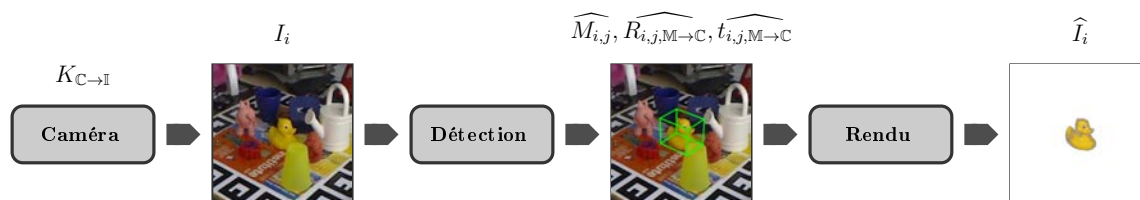


FIGURE 2.1 – Principe de fonctionnement d’une application de réalité mixte utilisant une caméra optique. La caméra fournit des images I_i en temps réel à l’application et ses paramètres intrinsèques K sont connus. Le module de détection 3D utilise ces images pour reconnaître les objets à augmenter $M_{i,j}$, leurs positions $t_{i,j}$ et leurs orientations $R_{i,j}$ ($\forall j$). Enfin un module d’affichage va mettre à jour la scène 3D virtuelle et générer les rendus correspondant aux images.

Puisque nous souhaitons développer une application en réalité mixte, la première question qui se pose à nous est :

Comment fonctionne une application de réalité mixte ?

Nous avons identifié deux techniques permettant d’afficher des informations en réalité mixte :

- Smartphone / Tablette : Les utilisateurs interagissent avec l’application par le biais d’un téléphone d’une tablette ou même simplement d’un PC équipé d’une caméra. Ce dernier affiche un rendu temps réel de la scène virtuelle par dessus un flux vidéo de la caméra filmant l’environnement réel à augmenter.
- Casque de réalité mixte : Les utilisateurs portent des lunettes ou un casque de réalité mixte. Ces dispositifs intègrent deux écrans semi-transparents qui permettent d’afficher un rendu stéréoscopique de la scène virtuelle devant les yeux de l’utilisateur.

Le matériel nécessaire à leur mise en œuvre est largement disponible aussi bien pour des applications personnelles que professionnelles. De plus, dans les deux cas l’application fonctionne selon le même principe : elle compose une scène virtuelle contenant les informations à afficher, dont elle calcule des rendus qui se superposent à la vue de l’utilisateur grâce à un ou plusieurs écrans. Pour que ces rendus se superposent correctement au flux vidéo de la caméra ou à la lumière qui arrive sur les écrans du casque de réalité mixte, l’application

calcule en temps réel les paramètres du système optique directement à partir des images de la caméra. Dans les casques de réalité mixte et les tablettes tactiles, des capteurs, tels que des centrales inertielles, sont aussi disponibles. Elles permettent de compléter les relevés du système optique, de mieux suivre les déplacements de l'utilisateur dans l'environnement à augmenter et donc, de positionner avec plus de précision les hologrammes. La figure 2.1 présente le fonctionnement général d'une application de réalité mixte selon ce principe.

Cette partie est dédiée à la présentation du fonctionnement d'un système de réalité mixte.

Dans un premier chapitre, nous présenterons le premier bloc du système dans la figure 2.1 : la caméra. Nous détaillerons le système optique mathématique qui explique la formation des images. C'est ce système qui définit les paramètres que l'application devra ensuite calculer en temps réel pour localiser l'utilisateur dans l'environnement, ou bien pour localiser les objets à augmenter par rapport à l'utilisateur.

Dans un deuxième chapitre, nous montrerons comment l'application peut aussi utiliser ces mêmes paramètres pour générer des rendus de la scène virtuelle, qui se superposent parfaitement aux images acquises par la caméra ce qui correspond au dernier bloc fonctionnel de l'application dans la figure 2.1.

Le troisième et dernier chapitre présente le bloc central de l'application dans la figure 2.1 : la détection des objets et l'estimation des paramètres qui leurs sont associés. Ce bloc est au centre de cette thèse, car comme nous allons le montrer dans cette partie, c'est un problème compliqué qui n'est pas encore entièrement résolu aujourd'hui.

2.1 Caméra et Images Réelles

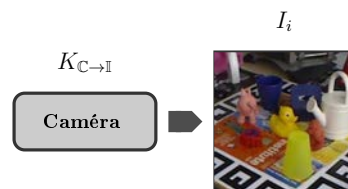


FIGURE 2.2 – Schéma illustrant l'acquisition d'une image par la caméra. La caméra optique produit des images I_i . La formation des images est régie par les propriétés "intrinsèques" de la caméra notées K .

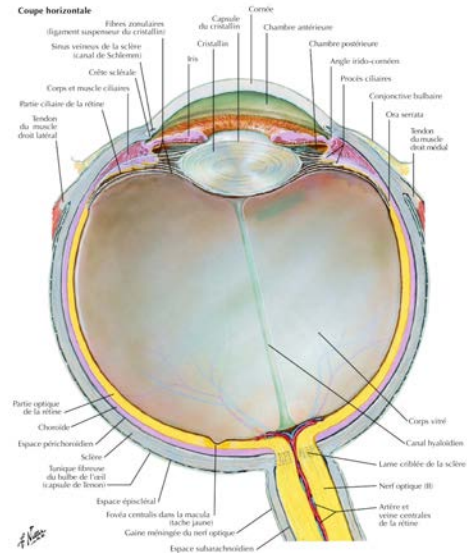
La caméra est LE capteur utilisé par notre application de réalité mixte pour fonctionner. C'est la première brique du système illustré en figures 2.1 et 2.2, dont toutes les autres dépendent. Plus précisément, c'est la formation des images qui nous intéresse. Pour pouvoir synthétiser une image contenant des informations à afficher en réalité mixte et les superposer à un flux vidéo, il va falloir répondre à cette question :

Comment la lumière en provenance des objets réels est-elle convertie en images par la caméra ?

Tout d'abord intéressons-nous à la structure interne d'une caméra, ou d'un œil humain pour le cas des applications utilisant un casque de réalité mixte.

Les caméras numériques, tout comme les caméras argentiques, utilisent une ou plusieurs lentilles pour focaliser la lumière qui rentre dans l'objectif. La lumière focalisée est ensuite convertie en signal électrique par un capteur photo-sensible. La figure 2.31 montre un appareil photo numérique qui fonctionne selon ce principe.

Dans le cas d'une application sur casque de réalité mixte, le système optique est comparable. Les écrans d'affichage sont placés devant les yeux de l'utilisateur. Les yeux sont



(1) Coupe d'un appareil photo Canon EOS ® dans laquelle les lentilles et le capteur photosensible sont visibles. Les lentilles sont les pièces en verre visible dans l'objectif et le zoom de l'appareil. Le capteur photosensible est la pièce colorée au centre de l'appareil photo.

(2) Planche 87 de l'Atlas d'anatomie humaine [83] illustrant l'œil humain. Le cristallin est une lentille biologique qui focalise la lumière perçue. Cette lumière est transformée en signal neuronal par la rétine correspondant au capteur photosensible.

FIGURE 2.3 – Comparaison entre une caméra et un œil. Bien que l'un soit biologique et l'autre électronique, les deux ont des composants qui remplissent des fonctions similaires et ils partagent le même système optique.

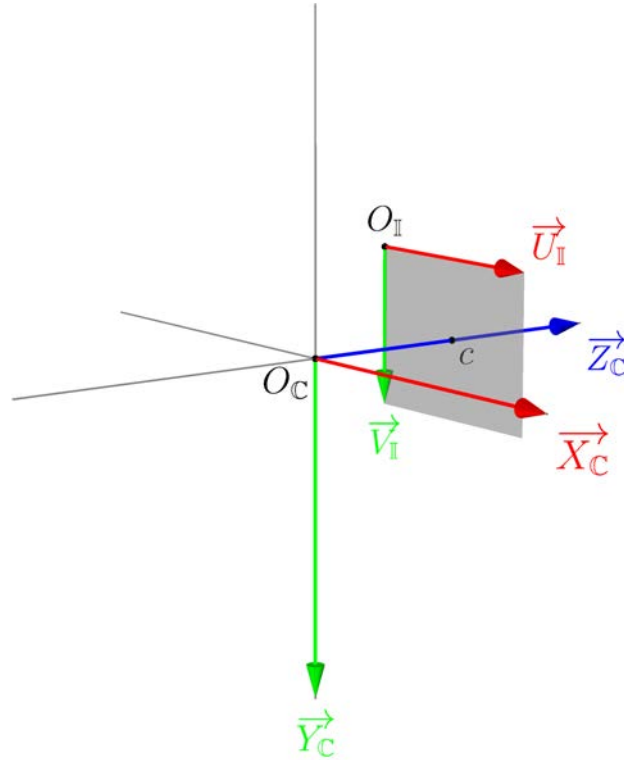
composés d'un cristallin et d'une rétine. Le cristallin joue le rôle d'une lentille et la rétine est un capteur photosensible biologique qui transforme la lumière perçue en signaux neuronaux. Un dessin de coupe de l'œil humain est présenté en figure 2.32.

Une caméra ou un œil, bien que différents, font intervenir des composants qui remplissent les mêmes fonctions. Ils peuvent donc être modélisés par le même système numérique. Ce système explique comment la lumière perçue depuis l'environnement est transformée en image.

2.1.1 Caméra et images

Pour modéliser le système optique de la caméra nous allons dans un premier temps définir le repère de la caméra et le repère image. Les composants internes de la caméra ainsi que les objets vus sont tous positionnés dans le repère de la caméra. Nous pourrions donc, dans un deuxième temps, définir leurs positions et leurs orientations dans ce repère. Cela nous permettra, dans une troisième partie, de modéliser numériquement le chemin de la lumière depuis les objets vus, jusqu'à l'image. Nous profiterons aussi de cette section pour présenter deux traitements d'images utilisés dans les caméras numériques et qui viennent compléter le système : le recadrage et le zoom. La première étape est de définir le repère 3D de la caméra que nous appellerons "C". Même si dans la réalité c'est la caméra qui se déplace dans l'environnement qu'elle filme, on considère ici que la caméra est au centre du système et que tous les objets sont placés par rapport à elle. C'est un système à la première personne, tout comme la perception visuelle humaine ou bien celle d'un drone. Le repère C est défini comme :

$$\mathbb{C} = \{O_C, \vec{X}_C, \vec{Y}_C, \vec{Z}_C\} \tag{2.1}$$


 FIGURE 2.4 – Répère de la caméra \mathbb{C} et du plan image \mathbb{I} .

Son origine est $O_C = (0m, 0m, 0m)$ et ses trois vecteurs unitaires sont les suivants :

$$\begin{aligned}\vec{X}_C &= \overrightarrow{(1m, 0m, 0m)} && \text{(de la gauche vers la droite)} \\ \vec{Y}_C &= \overrightarrow{(0m, 1m, 0m)} && \text{(du haut vers le bas)} \\ \vec{Z}_C &= \overrightarrow{(0m, 0m, 1m)} && \text{(de l'arrière vers l'avant)}\end{aligned}\tag{2.2}$$

L'axe \vec{Y}_C de ce repère est orienté vers le bas car ce repère est une extension en trois dimensions du repère image \mathbb{I} lui en deux dimensions qui est défini comme suit :

$$\mathbb{I} = \{O_I, \vec{U}_I, \vec{V}_I\}\tag{2.3}$$

L'origine du repère image \mathbb{I} est $O_I = (0px, 0px)$ et ses axes \vec{U}_I et \vec{V}_I sont orientés de la même façon que les axes \vec{X}_C et \vec{Y}_C du repère \mathbb{C} :

Le fait que l'axe vertical de ces deux repères soit orienté vers le bas est un héritage historique. En effet, les ordinateurs ont d'abord été capables d'afficher du texte et des images organisés en pages se lisant de haut en bas sur un écran. On a donc indexé les pixels des écrans d'ordinateurs dans le sens haut-bas. Les graphismes 3D sont arrivés plus tard sur ordinateur. Le modèle 2D a été étendu par l'ajout d'un troisième axe pour la profondeur, nommé Z . De nos jours l'affichage 3D comme extension de la 2D permet une rétro-compatible. Ainsi les outils 3D des ordinateurs permettent d'afficher des pages en 2D.

La caméra produit des images que nous noterons I_i où i est l'indice de la "i-ème" image. L'indice i est compris entre 0 inclu et $|I|$ exclu. $|I|$ est le nombre d'images acquises par la caméra. I dénote l'ensemble des images acquises par la caméra :

$$I = \{I_i \forall i \in [0, |I|]\}\tag{2.4}$$

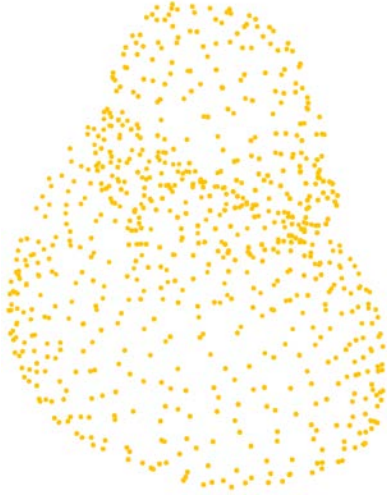


FIGURE 2.5 – Illustration d'un nuage de points obtenu par reprojection des images du "canard" du jeu de données LINEMOD. Pour la caméra, un objet n'est qu'un ensemble de points touché par des rayons de lumière. Ce nuage de points est ensuite re-maillé pour obtenir le modèle 3D de l'objet.

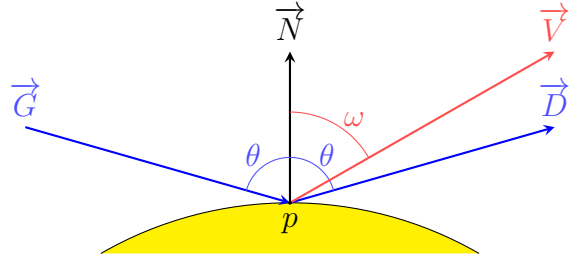


FIGURE 2.6 – Schéma expliquant le rebond de la lumière à la surface d'un objet selon le modèle de Phong [12, 124]. La lumière provient du côté gauche par le rayon \vec{G} . Elle éclaire l'objet jaune au point p . Si l'objet était un miroir parfait, la lumière serait réfléchi à droite par le rayon \vec{D} . Autrement, elle est diffusée dans toutes les directions en des rayons comme \vec{V} .

2.1.2 Objets 3D

k	$M_{i,j,\mathbb{M}}$		
	$M_{i,j,k,x,\mathbb{M}}$	$M_{i,j,k,y,\mathbb{M}}$	$M_{i,j,k,z,\mathbb{M}}$
0	-0.0391	0.0148	-0.0004
1	0.0171	0.0134	-0.0383
2	-0.0466	0.0116	-0.0278
3	-0.0442	-0.0001	-0.0000
4	-0.0434	0.0062	-0.0004
5	-0.0530	-0.0051	-0.0256
6	-0.0443	0.0156	-0.0231
7	-0.0372	-0.0016	0.0013
8	0.0044	-0.0326	-0.0235
9	-0.0344	0.0101	0.0013
10	-0.0503	0.0084	-0.0262
...
7911	0.0167	0.0319	-0,0039

FIGURE 2.7 – Extrait des points composant le modèle 3D du canard de LINEMOD. Ces points sont définis dans le repère local de l'objet, noté \mathbb{M} . Le point $k = 4$ a, par exemple, pour coordonnées $M_{i,j,4,\mathbb{M}} = (-0.0434, 0.0062, -0.0004)$. Ses coordonnées en x , y et z sont égales à $M_{i,j,4,x,\mathbb{M}} = -0.0434$, $M_{i,j,4,y,\mathbb{M}} = 0.0062$ et $M_{i,j,4,z,\mathbb{M}} = -0.0004$.

Des sources de lumière de l'environnement comme le soleil ou des lampes éclairent les objets. Sans lumière ils ne seraient pas visibles. Les rayons de lumière en provenance de ces sources touchent les objets. Les rayons étant modélisés comme des droites, ils touchent les objets en des points. Les objets vont réfléchir et éventuellement diffuser ces rayons, qui

seront ensuite perçus par la caméra. Les rayons réfléchis peuvent aussi aller éclairer d'autres objets. La caméra convertit les rayons qu'elle perçoit en image. Le modèle de propagation de la lumière a été caractérisé et modélisé par Phong [12, 124] qui travaillait sur les rendus photoréalistes. La figure 2.6 illustre le rebond de la lumière sur un objet selon ce modèle.

Les rayons de lumière en provenance d'un objet qui sont perçus par la caméra ont donc forcément rebondis sur cet objet en un certain nombre de points. Du point de vue de la caméra, cet objet est donc défini comme un ensemble de points, aussi appelé "nuage de points". Un objet sous la forme d'un nuage de points est illustré en figure 2.5.

Chaque image I_i peut présenter plusieurs objets. Nous noterons l'ensembles des objets présents dans cette image comme M_i :

$$M_i = \{ M_{i,j}, \forall j \in [0, |M_i| [] \} \quad (2.5)$$

Ici $M_{i,j}$ dénote un objet au sens "*nuage de points*". On notera $M_{i,j,k}$ le "*k-ème*" point de l'objet $M_{i,j}$. L'indice k est comprise entre 0 inclu et $|M_{i,j}|$ exclu. Où $|M_{i,j}|$ dénote le nombre de points dans le modèle $M_{i,j}$. Les points de l'objet sont des points 3D et nous noterons leurs trois coordonnées ($M_{i,j,k,x}$, $M_{i,j,k,y}$, $M_{i,j,k,z}$) :

$$M_{i,j,\mathbb{C}} = \{ M_{i,j,k,\mathbb{C}} \forall k \in [0, |M_{i,j}| [] \} \quad (2.6)$$

$$M_{i,j,k,\mathbb{C}} = (M_{i,j,k,x,\mathbb{C}}, M_{i,j,k,y,\mathbb{C}}, M_{i,j,k,z,\mathbb{C}}) \quad (2.7)$$

Les points du modèle sont pour le moment définis dans le repère de la caméra \mathbb{C} . Toutes les images présentant le même objet dans des positions et orientations différentes vont donc avoir un nuage de points différent du point de vue de la caméra. Il est peu pratique et très coûteux de stocker le nuage de points pour chaque image. Rappelons que le "canard" de LINEMOD est par exemple constitué de 7912 points, et le canard est présent dans 1254 images ! Hors nous savons que c'est le même objet qui est vu dans ces images, donc le même nuage de points, du point de vue de l'objet. Pour pouvoir travailler avec un seul nuage de points, nous allons le définir, et le recalculer, dans un repère local. Ce repère local sera noté \mathbb{M} . Il est identique à celui de la caméra : même axes et directions (c.f. Section 2.1.1 ci-dessus). On va donc préférer stocker les nuages de points dans le repère local, ce qui nous permet de n'en stocker qu'un par objet :

$$M_{i,j,\mathbb{M}} = \{ M_{i,j,k,\mathbb{M}} \forall k \in [0, |M_{i,j}| [] \} \quad (2.8)$$

$$M_{i,j,k,\mathbb{M}} = (M_{i,j,k,x,\mathbb{M}}, M_{i,j,k,y,\mathbb{M}}, M_{i,j,k,z,\mathbb{M}}) \quad (2.9)$$

La figure 2.7 présente un extrait des points du "*canard*" de LINEMOD dans son repère local. Elle illustre notamment la notation des points et de leurs coordonnées comme définie ci-dessus. On a donc maintenant un seul nuage de points à stocker par objet. Dans la section suivante nous allons voir comment positionner tout un nuage de points, i.e. un objet, par rapport à la caméra. Ce qui revient à résoudre le problème suivant :

$$M_{i,j,\mathbb{C}} = ? \times M_{i,j,\mathbb{M}} \quad (2.10)$$

2.1.3 Positionnement 3D

Le repère de la caméra \mathbb{C} est maintenant définie et nous savons que chaque objet $M_{i,j}$ est composé de points défini dans un repère local, propre à l'objet. Nous savons aussi que chaque image I_i présente des objets différents. L'ensemble des objets vus à l'image est noté M_i . Tout ce qui nous manque pour caractériser une image est la position et l'orientation relative des nuages de points des objets par rapport à la caméra.

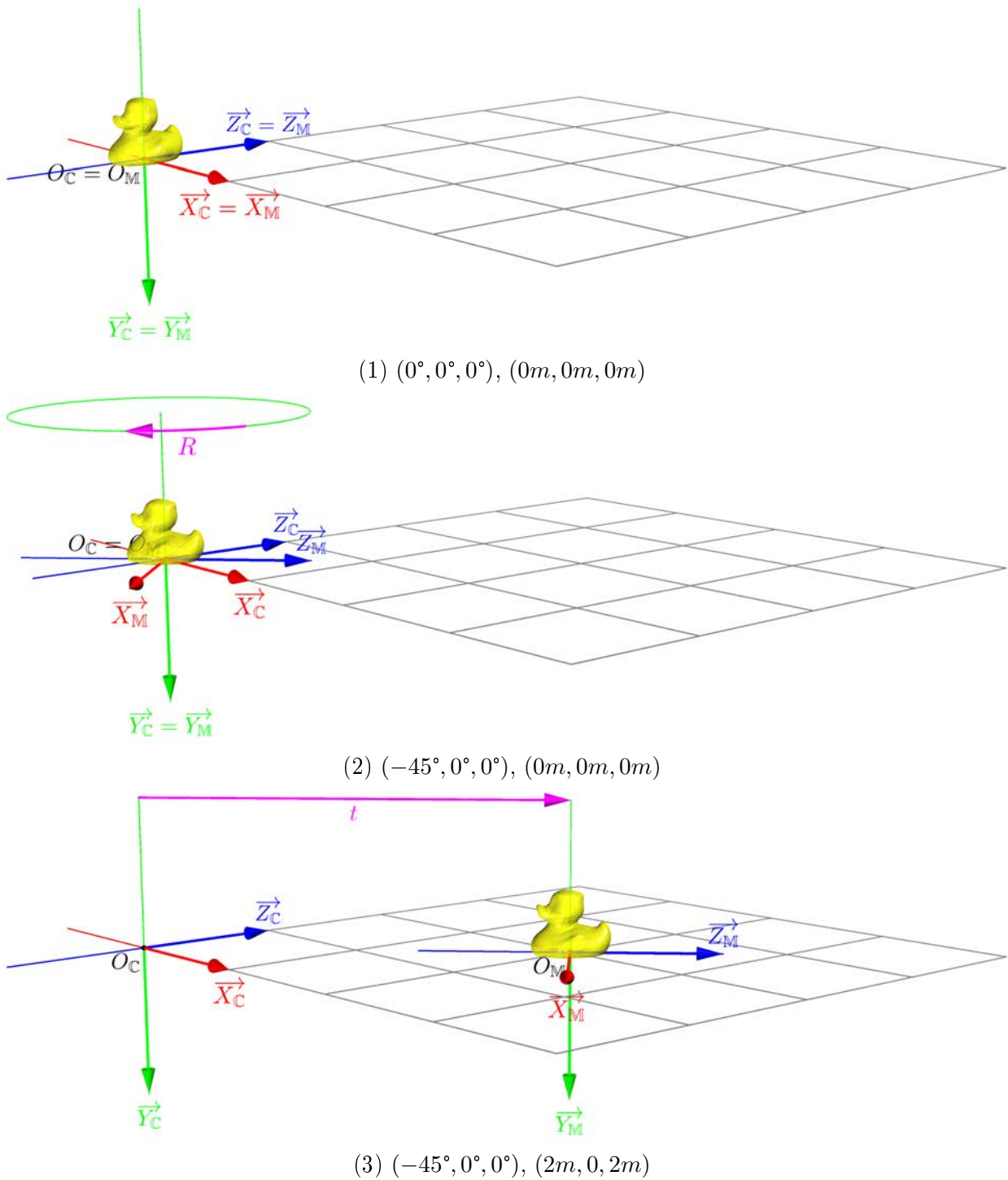


FIGURE 2.8 – Exemple de positionnement d’un objet dans le repère 3D de la caméra. La matrice de transformation affine $T_{i,j,M \rightarrow C}$ est ici décomposée en deux parties : sa partie rotation, exprimée en angles d’Euler (en degrés), et son vecteur de translation (en mètres). La rotation est appliquée avant la translation. Initialement, les deux repères sont identiques (a). La matrice de transformation affine $T_{i,j,M \rightarrow C}$ est alors égale à la matrice d’identité I . Puis, lorsque la rotation est appliquée (b), les directions des axes X_M , Y_M et Z_M peuvent être modifiées. Ici le canard est tourné d’un angle de -45° autour de l’axe Y_C , donc l’axe Y_M n’est pas modifié et reste égal à celui de la caméra Y_C . Ce sont les axes X_M et Z_M dont les directions sont affectées. Enfin la translation déplace tout le repère objet M par rapport au repère de la caméra C (c).

Nous noterons S_i la scène associée à l'image I_i . Chaque scène S_i présente un ensemble d'objets, et leurs positions et orientations respectives :

$$S_i = \{ M_{i,j}, R_{i,j,\mathbb{M} \rightarrow \mathbb{C}}, t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}, \forall j \in [0, |M_i|] \} \quad (2.11)$$

Le repère de la caméra \mathbb{C} est maintenant défini et nous savons que chaque objet $M_{i,j}$ est composé de points définis dans un repère local, propre à l'objet. Nous savons aussi que chaque image I_i présente des objets différents. L'ensemble des objets vus à l'image est noté M_i . Tout ce qui nous manque pour caractériser une image est la position et l'orientation relative des nuages de points des objets par rapport à la caméra.

La position de l'objet est caractérisée par le vecteur de translation 3D $t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ (en mètres) et sa rotation par la matrice de rotation 3D $R_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$. Ici nous supposons que les objets sont rigides et qu'un même objet n'existe qu'à une seule échelle. Ces deux paramètres sont nommés "paramètres extrinsèques" car ils ne dépendent pas de la caméra utilisée. Ils permettent de positionner l'objet par rapport à la caméra :

$$M_{i,j,\mathbb{C}} = [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{i,j,\mathbb{M}} \quad (2.12)$$

$$M_{i,j,\mathbb{C}} = T_{i,j,\mathbb{M} \rightarrow \mathbb{C}} \times M_{i,j,\mathbb{M}} \quad (2.13)$$

La position et orientation relative est aussi appelé "*pose*" d'un objet, comme si l'objet prenait la pose devant la caméra.

La lentille et le capteur photosensible sont positionnés de la même façon. On peut définir les valeurs de $R_{\mathbb{M} \rightarrow \mathbb{C}}$ et $t_{\mathbb{M} \rightarrow \mathbb{C}}$ pour ces deux composants. La lentille et le capteur photosensible sont alignés avec l'axe $\vec{Z}_{\mathbb{C}}$ et ils ne sont pas inclinés par rapport aux axes $X_{\mathbb{C}}$ et $Y_{\mathbb{C}}$. Leurs matrices de rotation sont donc égales à la matrice d'identité 3D. La lentille est positionnée à l'origine de la caméra, c'est elle qui définit le point d'entrée de la lumière dans le système optique. Sa translation $t_{\mathbb{M} \rightarrow \mathbb{C}}$ est donc nulle. Le capteur photosensible est aligné avec l'origine sur l'axe $\vec{Z}_{\mathbb{C}}$ de la caméra et est placé à une distance f de la lentille, qui est appelée distance focale. On a donc :

$$\begin{aligned} R_{\text{lentille},\mathbb{M} \rightarrow \mathbb{C}} &= I \\ t_{\text{lentille},\mathbb{M} \rightarrow \mathbb{C}} &= (0, 0, f) \\ R_{\text{capteur},\mathbb{M} \rightarrow \mathbb{C}} &= I \\ t_{\text{capteur},\mathbb{M} \rightarrow \mathbb{C}} &= (0, 0, 0) \end{aligned} \quad (2.14)$$

Notes :

- I est ici la matrice identité 3D, qui lorsqu'elle est utilisée comme matrice de rotation correspond à l'absence de rotation.
- Dans la réalité la distance f est négative car le capteur se trouve derrière la lentille, et de ce fait l'image apparaît à l'envers sur le plan image (le haut et le bas de l'image sont inversés). Pour obtenir une image dans le bon sens directement nous utilisons ici une valeur positive de f .

2.1.4 Formation des images

Une fois les objets positionnés par rapport à la caméra nous pouvons retracer le parcours de la lumière. Les rayons provenant des objets observés par la caméra sont focalisés à l'aide d'une lentille. Cette lentille a deux propriétés optiques majeures qui sont ses longueurs focales horizontales et verticales. Ces propriétés expriment dans quelle proportion la lentille distord

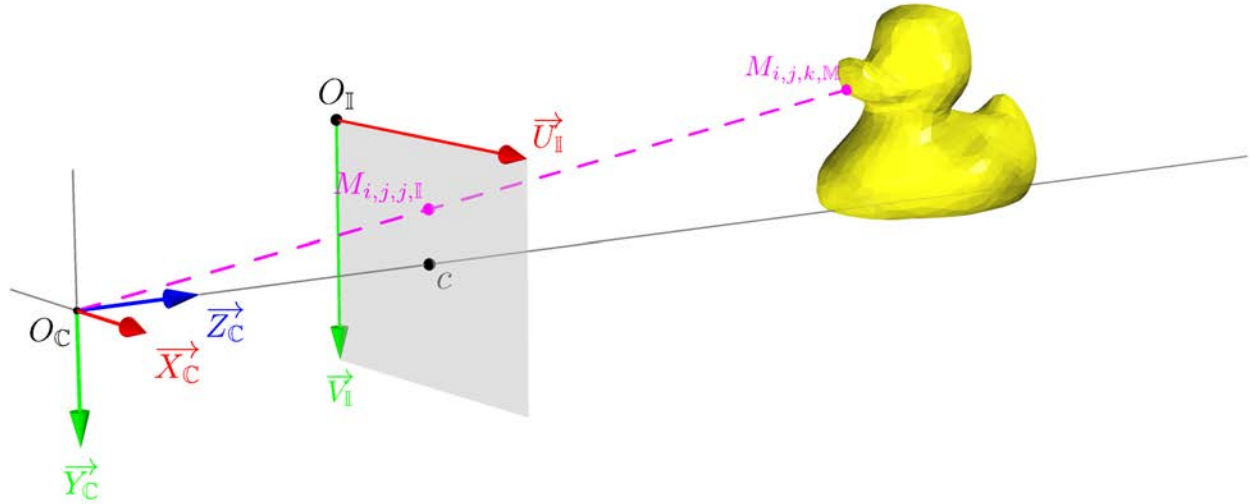


FIGURE 2.9 – Projection d'un point du modèle 3D dans le plan image suivant le parcours de la lumière. La ligne discontinue (en magenta) illustre le chemin suivi par la lumière. La lumière part du point $M_{i,j,k,M}$ au bout du bec du canard (définie dans le repère de l'objet). Elle termine sa course sur le plan image dans le capteur photo sensible au point $M_{i,j,k,I}$ (définie dans le repère image).

la lumière horizontalement et verticalement. Les rayons de lumière focalisés sont ensuite détectés par un capteur planaire constitué de cellules photosensibles disposées en grille. Ce capteur est caractérisé par sa résolution : le nombre de cellules photosensibles sur la largeur multiplié par le nombre de cellules sur sa hauteur. Ensemble, ces paramètres caractérisent la formation de l'image selon le modèle "trou d'épingle" ("pinhole" en anglais). Ils sont regroupés dans la matrice de "paramètres intrinsèques" notée K qui est propre à la caméra :

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.15)$$

Grâce aux paramètres intra et extrinsèques on peut calculer la projection d'un point appartenant au modèle 3D d'un objet dans le plan image :

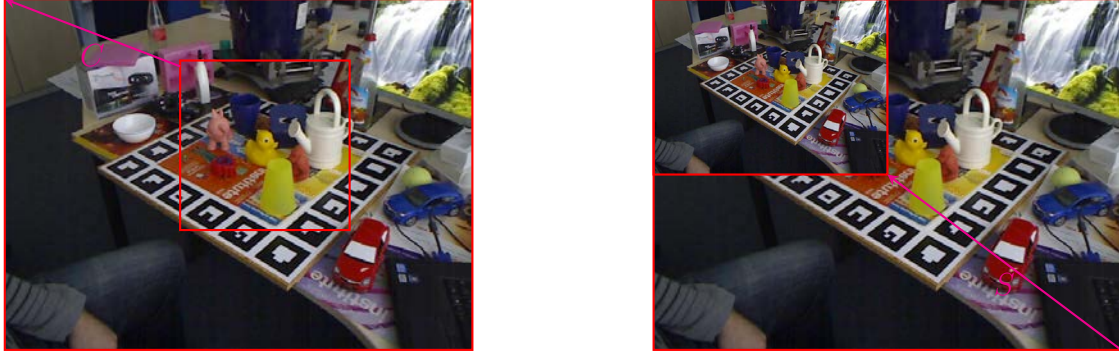
$$M_{i,j,k,I} = K_{C \rightarrow I} \times [R_{i,j,M \rightarrow C} \mid t_{i,j,M \rightarrow C}] \times M_{i,j,k,M} \quad (2.16)$$

Une fois ce calcul appliqué à tous les points de tous les objets on obtient l'image :

$$I_i = \bigcup_{j=0}^{|M_i|} K_{C \rightarrow I} \times [R_{i,j,M \rightarrow C} \mid t_{i,j,M \rightarrow C}] \times M_{i,j} \quad (2.17)$$

Note :

- $\bigcup_{j=0}^{|M_i|}$ indique que l'on calcule la projection des points de tous les modèles 3D pour obtenir l'image. Dans le cas où plusieurs points se superposent, seul le point le plus proche de la caméra sera visible. Sauf si il est partiellement ou complètement transparent, auquel cas, un mixage des couleurs des points est opéré.



(1) Exemple de recadrage d'une image. La transformation affine C change l'origine de l'image selon la flèche magenta. Les coordonnées des points définis dans le repère image sont traduites par C .

(2) Exemple de zoom sur une image. La transformation affine S réduit la taille de l'image par 2. Les coordonnées des points définis dans le repère image sont donc divisées par 2.

FIGURE 2.10 – Illustrations du recrage et de la mise à l'échelle d'une image.

2.1.5 Récapitulatif

Le point 3D $M_{i,j,k,\mathbb{M}}$ appartenant au " j -ème" modèle 3D vu dans la " i -ème" image est tout d'abord amené dans le repère 3D de la caméra par multiplication avec la matrice de transformation affine $[R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}]$. Il est ensuite projeté en deux dimensions sur le plan image par multiplication avec la matrice de projection perspective K . On obtient donc un point $M_{i,j,k,\mathbb{I}}$ à la surface du plan image. La figure 2.9 illustre graphiquement le résultat de ce calcul.

2.1.6 Recadrage

Les caméras fournissent des options comme le recadrage numérique. Ce recadrage est aussi utilisé comme pré-traitement des images pour l'entraînement des réseaux de neurones. Il est donc important d'inclure cette option dans la formulation du système optique. Le recadrage intervient après la formation de l'image. Cette opération, que nous noterons C comme "crop" en anglais, vient s'adjoindre à gauche de l'Équation 2.16 et modifie les coordonnées des points dans le plan image :

$$M_{i,j,k,\mathbb{I}} = C \times M_{i,j,k,\mathbb{I}} \quad (2.18)$$

$$M_{i,j,k,\mathbb{I}} = C \times K_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{i,j,k,\mathbb{M}} \quad (2.19)$$

$$I_i = \bigcup_{j=0}^{|M_i|} C \times K_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{i,j} \quad (2.20)$$

La transformation affine C est une matrice de translation à deux dimensions qui décale les points à la surface de l'image :

$$C = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.21)$$

La figure 2.101 illustre l'application de la transformation affine C sur une image du jeu de données LINEMOD. La transformation affine C et la matrice de projection perspective K peuvent se combiner par multiplication. Il en résulte une nouvelle matrice de paramètres

intrinsèques K' incluant l'opération de recadrage. Cette matrice pourra être utilisée partout où la matrice K apparaît :

$$K'_{\mathbb{C} \rightarrow \mathbb{I}} = C \times K_{\mathbb{C} \rightarrow \mathbb{I}} \quad (2.22)$$

2.1.7 Zoom

Le zoom numérique augmente ou réduit la taille d'une image. De ce fait, le nombre de pixels après le zoom est modifié et les positions des points projetés depuis les objets changent également. De même, au lieu de positionner les points sur le point image par leurs coordonnées en pixels, il est possible d'utiliser une représentation normalisée du plan image. En coordonnée normalisée le point $(0, 0)$ est par exemple est le coin supérieur gauche de l'image ; le point $(1, 1)$ est lui le coin inférieur droit (et non pas le second pixel de la diagonale) ; et le point $(0.5, 0.5)$ est au milieu de l'image. Ce changement d'échelle s'effectue par la même opération mathématique que le zoom. Le zoom est un changement d'échelle de tous les points sur l'image. C'est donc une opération, que nous noterons S comme "scaling" en anglais, qui vient s'adjoindre à gauche de l'Équation 2.16 et qui modifie les coordonnées des points dans le plan image :

$$M_{i,j,k,\mathbb{I}'} = S \times M_{i,j,k,\mathbb{I}} \quad (2.23)$$

$$M_{i,j,k,\mathbb{I}'} = S \times K_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{i,j,k,\mathbb{M}} \quad (2.24)$$

$$I_i = \bigcup_{j=0}^{|M_i|} C \times K_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{i,j} \quad (2.25)$$

La transformation affine S est une matrice de mise à l'échelle à deux dimensions qui met à l'échelle les points à la surface l'image selon l'axe horizontale $\vec{U}_{\mathbb{I}}$ et vertical $\vec{V}_{\mathbb{I}}$:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.26)$$

La figure 2.102 illustre l'application de la transformation affine S sur une image du jeu de données LINEMOD. La transformation affine S peut être combinée avec la matrice de projection perspective K en les multipliant l'une à l'autre pour obtenir une nouvelle matrice paramètre intrinsèque K' incluant l'option de mise à l'échelle. Cette matrice pourra être utilisée partout où la matrice K apparaît, comme pour le recadrage :

$$K'_{\mathbb{C} \rightarrow \mathbb{I}} = S \times K_{\mathbb{C} \rightarrow \mathbb{I}} \quad (2.27)$$

Conclusion

Dans cette section nous avons donc vu comment une caméra optique numérique capte la lumière et la transforme en image en deux dimensions. Ce processus peut se décrire mathématiquement par une suite de transformations affines et une projection perspective faisant intervenir deux groupes de paramètres : ceux qui dépendent seulement de la caméra et décrivent ses propriétés optiques, et ceux qui décrivent les positions des objets par rapport à la caméra dans chaque image. Les paramètres intrinsèques sont notés K et les paramètres extrinsèques $\{R_{i,j,\mathbb{M} \rightarrow \mathbb{C}}, t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}\}$. De plus, nous avons vu comment le zoom numérique et le recadrage affectent les points de l'image et comment combiner ces opérations directement avec la projection perspective dans la matrice K .

Dans la prochaine section nous allons voir comment utiliser ces paramètres pour créer des rendus qui se superposent aux images de la caméra.

2.2 Rendus 3D

Introduction

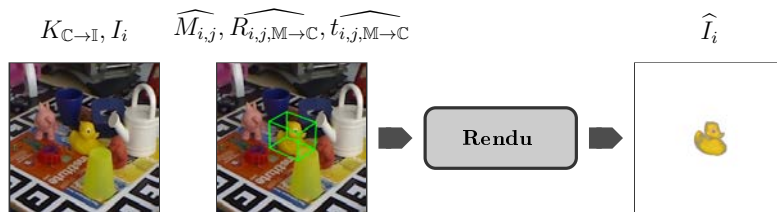


FIGURE 2.11 – Schéma illustrant la synthèse d’images. Les modèles 3D $M_{i,j}$, leurs positions $t_{i,j,M \rightarrow C}$ et leurs orientations $R_{i,j,M \rightarrow C}$ sont utilisés pour synthétiser le rendu \widehat{I}_i l’image I_i .

Nous avons vu comment les images sont acquises par la caméra et quels sont les paramètres impliqués. Imaginons que nous connaissons tous ces paramètres. Nous allons maintenant nous intéresser au problème suivant :

*Comment reproduire les images de la caméra
ou en produire de nouvelles qui pourront s’y superposer ?*

Pour ce faire, nous allons utiliser les techniques d’affichage 3D OpenGL moderne [106, 99, 100]. Comme mentionné dans l’introduction, ces techniques, couplées à un casque de réalité mixte ou à une tablette, permettent de réaliser directement des applications de réalité mixte. Elles permettent aussi de générer des données d’annotation qui peuvent servir à entraîner des réseaux de neurones à détecter les objets. Comme, par exemple, des masques de segmentation ou des cartes de profondeur.² Les rendus peuvent aussi être utilisés en comparaison aux images acquises pour affiner l’inférence des paramètres extrinsèques.

Pour générer des rendus, les modèles 3D et une caméra virtuelle sont placés dans une scène virtuelle qui a pour but de reproduire les conditions d’acquisition des images prises avec une caméra physique : la disposition des objets, les propriétés physiques de la caméra et le parcours de la lumière sont reproduits.

La figure 2.12 présente une scène 3D complète. Tous les repères 3D utilisés suivent la norme d’OpenGL que nous utilisons pour l’implémentation. Les sous-sections suivantes présentent les différents repères qui composent la scène et comment passer numériquement de l’un à l’autre. L’objectif étant d’arriver à une image 2D en partant d’un objet 3D et des paramètres de pose associés.

2.2.1 Scène 3D

La scène, notée \mathbb{S} , est illustrée en figure 2.13. C’est un objet virtuel qui n’apparaît pas à l’image. La scène définit un repère orthonormé dont l’unité est le mètre et dans lesquels tous les modèles 3D et la caméra sont placés :

$$S = \{O_S, \vec{X}_S, \vec{Y}_S, \vec{Z}_S\} \quad (2.28)$$

Son origine est $O_S = (0m, 0m, 0m)$ et ses trois vecteurs unitaires sont :

- $\vec{X}_S = \overrightarrow{(1m, 0m, 0m)}$: de la gauche vers la droite.
- $\vec{Y}_S = \overrightarrow{(0m, 1m, 0m)}$: du bas vers le haut.

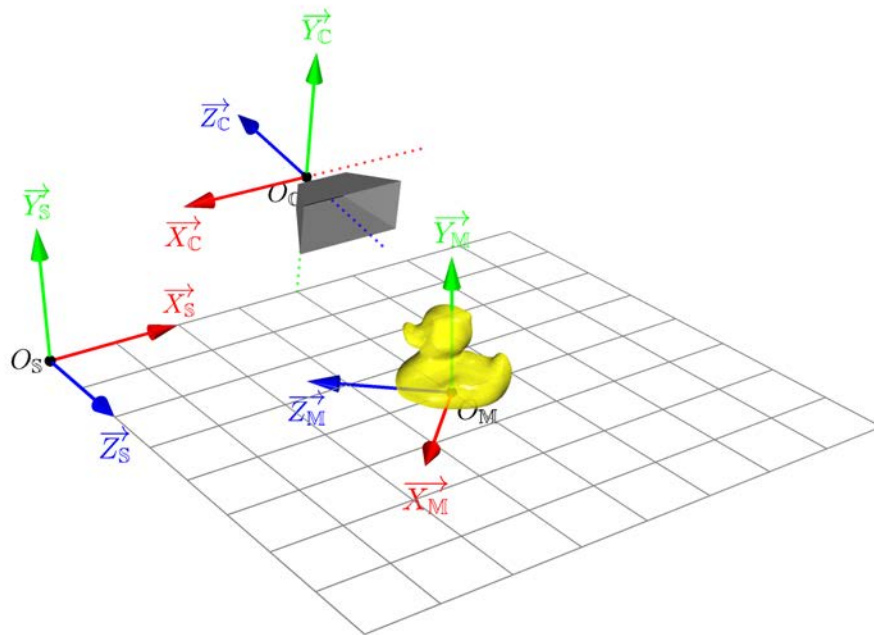


FIGURE 2.12 – Illustration des différents repères 3D utilisés pour l'affichage 3D afin de synthétiser des images. La caméra \mathbb{C} et le modèle 3D de canard \mathbb{M} sont placés dans la scène \mathbb{S} .

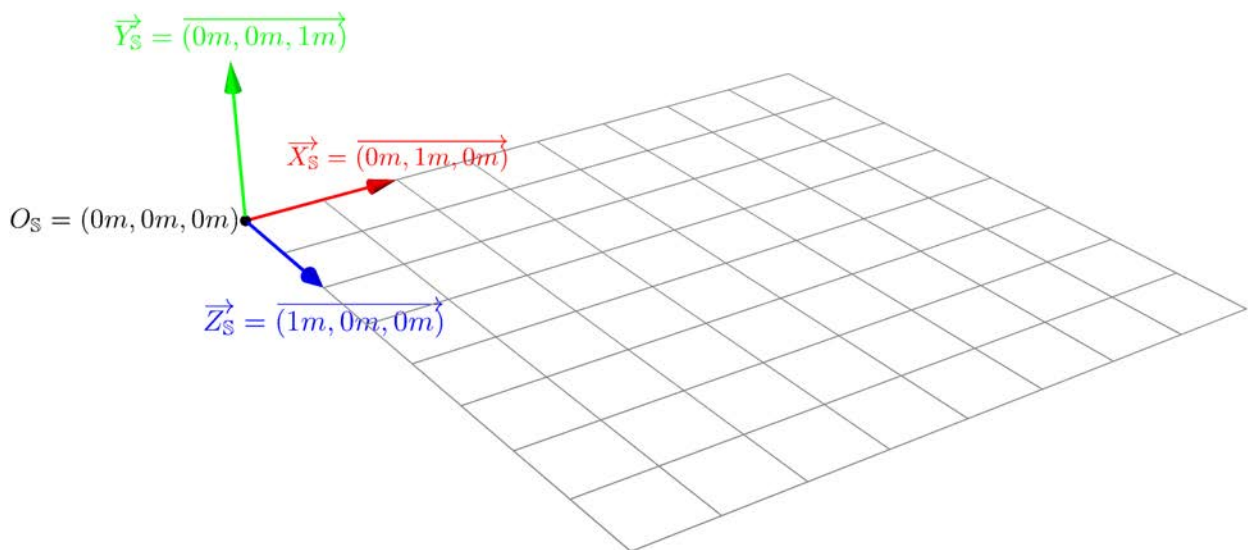


FIGURE 2.13 – Repère de la scène \mathbb{S} dans lequel sont placés tous les objets, y compris la caméra.

- $\vec{Z}_S = \overrightarrow{(0m, 0m, 1m)}$: de l'avant vers l'arrière.

Nous expliquerons pourquoi les directions et les sens des axes ont été choisis de cette façon dans la sous-section ci-dessous consacrée à la caméra.

2.2.2 Modèles 3D

La géométrie d'un modèle 3D est définie par un maillage triangulaire : des points en 3 dimensions sont reliés par des arêtes. Chaque triplé d'arêtes détoure une face triangulaire. Cette géométrie est définie dans un repère local comme présenté en figure 2.14. Ce repère \mathbb{M} est défini de la même façon que celui de la scène (c.f. Équation 2.28) :

$$\mathbb{M} = \{O_M, \vec{X}_M, \vec{Y}_M, \vec{Z}_M\} \quad (2.29)$$

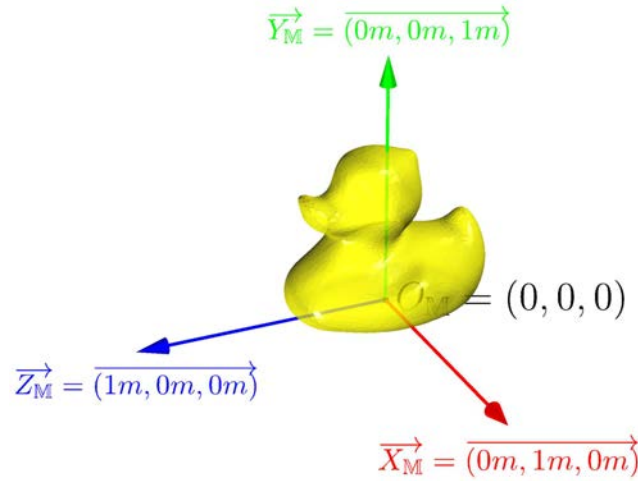


FIGURE 2.14 – Repère objet $\mathbb{M} = \{O, \vec{X}, \vec{Y}, \vec{Z}\}$ dans lequel les points des modèles 3D sont définis.

Il est, lui aussi, centré sur l'origine $O_{\mathbb{M}} = (0m, 0m, 0m)$ et ses trois vecteurs unitaires sont aussi :

- $\vec{X}_{\mathbb{M}} = \overrightarrow{(1m, 0m, 0m)}$: de la gauche vers la droite.
- $\vec{Y}_{\mathbb{M}} = \overrightarrow{(0m, 1m, 0m)}$: du bas vers le haut.
- $\vec{Z}_{\mathbb{M}} = \overrightarrow{(0m, 0m, 1m)}$: de l'avant vers l'arrière.

Comme dans la section précédente, la géométrie du modèle $M_{i,j}$ est définie comme l'ensemble des points 3D vus par la caméra.

Tous les logiciels 3D n'utilisent pas nécessairement la norme OpenGL pour l'orientation du repère objet. Certains laissent la possibilité à l'utilisateur de choisir le repère utilisé pour sauvegarder le modèle 3D, mais ce n'est pas toujours le cas. Par exemple, OpenCV partage le même axe X qu'OpenGL, mais les directions des axes Y et Z sont inversées. Le repère utilisé pour le modèle 3D est généralement le même utilisé pour définir les paramètres extrinsèques associés à l'objet ($R_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ et $t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$). Il est donc essentiel d'effectuer le positionnement de l'objet dans le repère dans lequel il est défini, et si nécessaire, de changer ensuite de repère pour revenir à la norme OpenGL que nous utilisons pour calculer les rendus. Nous verrons comment inclure le changement de repère dans la sous-section ci-dessous consacrée au positionnement des objets.

2.2.3 Caméra

La caméra est un objet à part. Elle est placée dans la scène S , mais n'apparaît pas à l'image. Encore une fois, le repère utilisé par la caméra est identique à celui de la scène S mais nommé \mathbb{C} (c.f. Équation 2.28) :

$$\mathbb{C} = \{O_{\mathbb{C}}, \vec{X}_{\mathbb{C}}, \vec{Y}_{\mathbb{C}}, \vec{Z}_{\mathbb{C}}\} \quad (2.30)$$

Son origine est $O_{\mathbb{S}} = (0m, 0m, 0m)$ et ses trois vecteurs unitaires sont :

- $\vec{X}_{\mathbb{C}} = \overrightarrow{(1m, 0m, 0m)}$: de la gauche vers la droite.
- $\vec{Y}_{\mathbb{C}} = \overrightarrow{(0m, 1m, 0m)}$: du bas vers le haut.
- $\vec{Z}_{\mathbb{C}} = \overrightarrow{(0m, 0m, 1m)}$: de l'avant vers l'arrière.

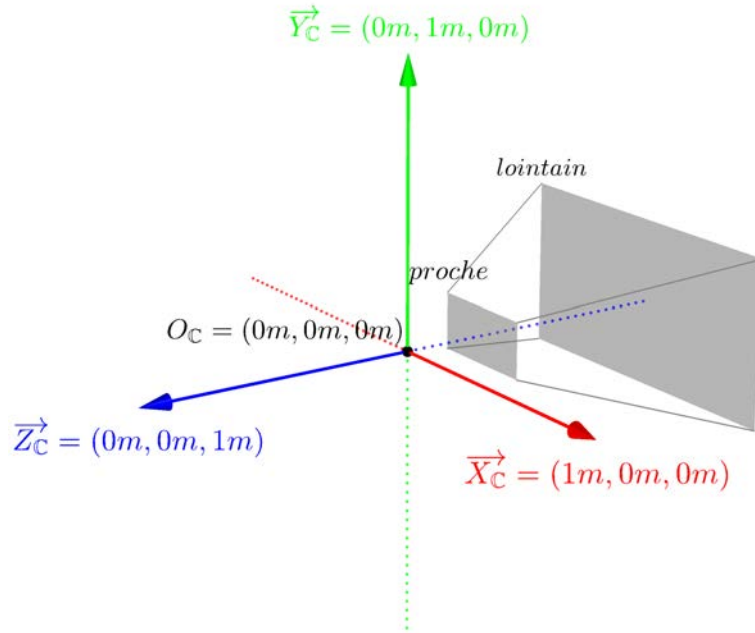


FIGURE 2.15 – Repère 3D de la caméra \mathcal{C} . Les objets sont d’abord projetés dans ce repère avant d’être dessinés dans le plan image. Seuls les objets compris dans le trapèzoïde délimité par les plans de coupures proches (near) et lointains (far) seront visibles à l’image. Ces plans de coupures correspondent aux distances minimum et maximum de vue. La largeur et la hauteur du trapèzoïde dépendent de la taille de l’image dessinée et de l’angle de vue utilisé.

Pour calculer les rendus, tous les objets visibles sont transposés dans le repère 3D de la caméra, puis projetés dans le plan image en suivant le parcours de la lumière pour créer les rendus. C’est pourquoi les axes de ce repère sont définis par analogie avec le plan image et le chemin que suit la lumière pour arriver jusqu’à l’image :

- L’axe \vec{X}_C va de la gauche vers la droite car c’est le sens naturel de lecture d’une image.
- L’axe \vec{Y}_C va du bas vers le haut car en 3D on ne cherche pas à visualiser du texte qui se lit de haut en bas, mais des images qui seront perçues du bas vers le haut.
- L’axe \vec{Z}_C va de l’avant vers l’arrière car c’est le sens du parcours de la lumière qui part des objets en avant de la caméra et qui se dirige vers la caméra.

2.2.4 Positionnement des objets

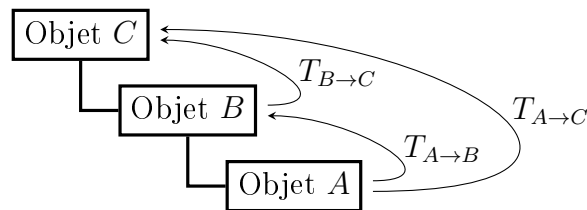


FIGURE 2.16 – Exemple d’une hiérarchie d’objets et des transformations affines permettant de passer du repère local de l’un à celui d’un autre.

Une transformation affine T est une matrice de dimensions 4×4 . Lorsque l’on multiplie cette matrice à un point 3D à coordonnées homogènes $(x, y, z, 1)$, on obtient un nouveau

point qui est translaté et tourné selon la transformation T :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.31)$$

$$\iff \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} T_{0,0} & T_{1,0} & T_{2,0} & T_{3,0} \\ T_{0,1} & T_{1,1} & T_{2,1} & T_{3,1} \\ T_{0,2} & T_{1,2} & T_{2,2} & T_{3,2} \\ T_{0,3} & T_{1,3} & T_{2,3} & T_{3,3} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.32)$$

$$\iff \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} T_{0,0} \times x & T_{1,0} \times y & T_{2,0} \times z & T_{3,0} \times 1 \\ T_{0,1} \times x & T_{1,1} \times y & T_{2,1} \times z & T_{3,1} \times 1 \\ T_{0,2} \times x & T_{1,2} \times y & T_{2,2} \times z & T_{3,2} \times 1 \\ T_{0,3} \times x & T_{1,3} \times y & T_{2,3} \times z & T_{3,3} \times 1 \end{bmatrix} \quad (2.33)$$

La position du repère local de l'objet A par rapport au repère local de l'objet B peut être définie par une matrice de transformation affine $T_{A \rightarrow B}$. En effet, cette transformation affine décrit ce changement de repère comme la composition de deux opérations. Une rotation $R_{A \rightarrow B}$ et une translation $t_{A \rightarrow B}$:

$$T_{A \rightarrow B} = [R_{A \rightarrow B} \mid t_{A \rightarrow B}] \quad (2.34)$$

Il est aussi possible de définir par la suite la position de l'objet B par rapport à un objet C grâce à la transformation affine $T_{B \rightarrow C}$. La position de l'objet A par rapport à l'objet C est alors définie par la transformation affine $T_{A \rightarrow C}$ qui est la composition des transformations affines $T_{A \rightarrow B}$ et $T_{B \rightarrow C}$ et qui correspond à la hiérarchie illustrée à la figure 2.16 :

$$T_{A \rightarrow C} = T_{A \rightarrow B} \times T_{B \rightarrow C} \quad (2.35)$$

Considérons maintenant que l'on souhaite générer le rendu de la scène S_i associée à la " i -ème" image acquise par la caméra (voire Chapitre 2.1). On rappelle que cette scène présente un ensemble d'objet noté M_i . Chaque objet vu, noté $M_{i,j}$, est alors positionné et orienté par rapport à la scène grâce à une matrice de transformation affine $T_{i,\mathbb{M} \rightarrow \mathbb{S}}$ qui décrit mathématiquement les opérations de rotation et de translation qui permettent de passer du repère du modèle \mathbb{M} au repère de la scène \mathbb{S} :

$$T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} = [R_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \mid t_{i,j,\mathbb{M} \rightarrow \mathbb{S}}] \quad (2.36)$$

Cette matrice de transformation affine de dimension 4×4 s'applique sur des coordonnées homogènes. Par exemple pour les sommets du modèle 3D : le point $(M_{i,j,k,x}, M_{i,j,k,y}, M_{i,j,k,z})$ à pour coordonnée homogène $(M_{i,j,k,x}, M_{i,j,k,y}, M_{i,j,k,z}, 1)$. L'application du passage du repère du modèle 3D à celui de la scène 3D se fait en multipliant les coordonnées homogènes des sommets du modèle 3D par la transformation affine :

$$M_{i,j,k,\mathbb{S}} = T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \times M_{i,j,k,\mathbb{M}} \quad (2.37)$$

$$\iff \begin{bmatrix} M_{i,j,k,x,\mathbb{S}} \\ M_{i,j,k,y,\mathbb{S}} \\ M_{i,j,k,z,\mathbb{S}} \\ 1 \end{bmatrix} = T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \times \begin{bmatrix} M_{i,j,k,x,\mathbb{M}} \\ M_{i,j,k,y,\mathbb{M}} \\ M_{i,j,k,z,\mathbb{M}} \\ 1 \end{bmatrix} \quad (2.38)$$

Comme mentionné ci-dessus en section 2.2.2 la matrice de rotation $R_{i,j,\mathbb{M} \rightarrow \mathbb{S}}$ et le vecteur de translation $t_{i,j,\mathbb{M} \rightarrow \mathbb{S}}$, et donc la matrice de transformation affine $T_{i,j,\mathbb{M} \rightarrow \mathbb{S}}$, doivent être

définis dans le même repère que celui choisi pour décrire la géométrie du modèle 3D $M_{i,j}$ et la scène S_i . Si ce n'est pas le cas, la rotation et la translation se feront selon les mauvais axes, ou du moins, pas ceux souhaités. C'est le cas, par exemple, de plusieurs jeux de données où les modèles 3D et leurs poses sont obtenus à l'aide de la bibliothèque OpenCV. Dans ce cas, la matrice de transformation affine $T_{i,j,M \rightarrow S}$ amène le modèle 3D OpenCV de l'objet dans une scène dont les axes sont ceux de la norme OpenCV et non pas OpenGL. Ils faut donc passer de la scène OpenCV à la scène OpenGL avant de passer dans le repère de la caméra OpenGL. Sinon, l'objet ne sera pas dessiné dans le bon sens. Ce passage se fait grâce à une autre matrice de transformation affine $T_{CV \rightarrow GL}$ qui inverse la direction des axes Y et Z du repère :

$$T_{CV \rightarrow GL} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.39)$$

Si on reprend l'équation 2.37 et que l'on tient compte du changement de repère qui a lieu après la rotation et la translation, on a alors l'équation :

$$M_{i,j,k,S} = T_{CV \rightarrow GL} \times T_{i,j,M \rightarrow S} \times M_{i,j,k,M} \quad (2.40)$$

2.2.5 Positionnement de la caméra

De la même façon que pour les objets, la caméra est placée dans la scène. La matrice de transformation affine $T_{i,C \rightarrow S}$ permet de positionner la caméra dans la scène exactement comme les autres objets :

$$T_{i,C \rightarrow S} = [R_{C \rightarrow S} | t_{C \rightarrow S}] \quad (2.41)$$

Comme mentionné dans la sous-section 2.2.3 ci-dessus consacrée au repère 3D de la caméra \mathbb{C} , les rendus sont obtenus en projetant tous les objets dans le repère \mathbb{C} , puis dans le plan image \mathbb{I} . C'est donc la matrice de transformation affine $T_{i,S \rightarrow C}$, permettant de transformer les points définis dans la scène 3D en point dans le repère de la caméra 3D, qui est d'importance pour calculer les rendus. Cette matrice est tout simplement l'inverse de la matrice $T_{i,C \rightarrow S}$:

$$T_{i,S \rightarrow C} = (T_{i,C \rightarrow S})^{-1} \quad (2.42)$$

Nous savons donc maintenant comment passer du repère 3D d'un objet au repère 3D de la scène, puis de la scène à celui de la caméra 3D, et donc directement du repère de l'objet à celui de la caméra. Nous allons maintenant voir comment projeter les modèles 3D sur le plan image.

2.2.6 Projection dans le plan image

Pour obtenir des rendus il ne nous reste plus qu'à passer sur le plan image. Ce "passage" s'effectue en projetant les points des modèles 3D dans le plan image grâce à une matrice de projection perspective nommée $P_{C \rightarrow \mathbb{I}}$:

$$P_{C \rightarrow \mathbb{I}} = \begin{bmatrix} 2K_{0,0}/w & -2K_{0,1}/w & (-2K_{0,2} + w + 2x_0)/w & 0 \\ 0 & 2K_{1,1}/h & (2K_{1,2} + h + 2y_0)/h & 0 \\ 0 & 0 & -(fc + nc)/(fc - nc) & -2(fc * nc)/(fc - nc) \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (2.43)$$

Où K est la matrice de paramètres intrinsèques telle que définie à la sous-section 2.1.4 consacrée au chemin de la lumière. Puisque l'on cherche à obtenir des images discrètes (composées de pixels), la dimension du plan image \mathbb{I} est exprimée en pixels par sa largeur w et sa hauteur h . Le plans image \mathbb{I} peut être décalé horizontalement de x_0 pixels et verticalement de y_0 pixels notamment pour recadrer le rendu. De plus, les profondeurs de champs minimales et maximales exprimées en mètres sont définies par nc ("near clip" en anglais) et fc ("far clip" en Anglais). Seuls les objets placés à une profondeur comprise entre ces deux valeurs sont rendus, autrement, on considère qu'ils ne sont pas visibles.

Cette matrice de projection perspective s'applique sur les coordonnées homogènes des modèles 3D dans le repère 3D de la caméra \mathbb{C} et les transforme en coordonnées sur le plan image \mathbb{I} en pixels :

$$M_{i,j,k,\widehat{\mathbb{I}}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times M_{i,j,k,\mathbb{C}} \quad (2.44)$$

$$\widehat{I}_i = \bigcup_{j=0}^{|M_i|} P_{\mathbb{C} \rightarrow \mathbb{I}} \times M_{i,j,\mathbb{C}} \quad (2.45)$$

2.2.7 Formulation Complète

En somme, pour calculer le rendu de la scène 3D S_i il faut transformer tous les points $p_{M_{i,j}}$ des modèles 3D en point $p_{\mathbb{C}}$ dans le repère de la caméra 3D \mathbb{C} et finalement projeter ces derniers dans le plan image \widehat{I}_i en des points 2D $p_{\widehat{I}_i}$. Ce calcul, qui est appliqué à chaque instance $M_{i,j}$ de modèle 3D, est donc la composition de toutes les étapes vues ci-dessus :

$$M_{i,j,k,\widehat{\mathbb{I}}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times T_{i,\mathbb{S} \rightarrow \mathbb{C}} \times T_{\mathbb{C}\mathbb{V} \rightarrow \mathbb{G}\mathbb{L}} \times T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \times M_{i,j,k,\mathbb{M}} \quad (2.46)$$

$$M_{i,j,k,\widehat{\mathbb{I}}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times (T_{i,\mathbb{C} \rightarrow \mathbb{S}})^{-1} \times T_{\mathbb{C}\mathbb{V} \rightarrow \mathbb{G}\mathbb{L}} \times T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \times M_{i,j,k,\mathbb{M}} \quad (2.47)$$

$$\widehat{I}_i = \bigcup_{j=0}^{|M_i|} P_{\mathbb{C} \rightarrow \mathbb{I}} \times (T_{i,\mathbb{C} \rightarrow \mathbb{S}})^{-1} \times T_{\mathbb{C}\mathbb{V} \rightarrow \mathbb{G}\mathbb{L}} \times T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \times M_{i,j,\mathbb{M}} \quad (2.48)$$

La transformation affine $T_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ est aussi souvent utilisée pour décrire directement la relation entre l'objet et la caméra. Cette transformation est appelée "matrice de vue" car elle oriente et positionne l'objet tel que vu dans l'image I_i et c'est notamment cette matrice qui apparaissait dans le système formulé au chapitre précédent consacré à la formation des images optique :

$$T_{i,j,\mathbb{M} \rightarrow \mathbb{C}} = T_{i,\mathbb{S} \rightarrow \mathbb{C}} \times T_{i,j,\mathbb{M} \rightarrow \mathbb{S}} \quad (2.49)$$

Et si on lui multiplie la matrice de projection $P_{\mathbb{C} \rightarrow \mathbb{I}}$, on obtient alors une nouvelle matrice de projection qui est la composée de tout le système optique de rendu :

$$P_{i,j,\mathbb{M} \rightarrow \mathbb{I}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times T_{i,j,\mathbb{M} \rightarrow \mathbb{C}} \quad (2.50)$$

Ces deux matrices sont utilisées dans notre programme de rendus 3D OpenGL.

Conclusion

Dans cette section nous avons vu comment générer des rendus à partir des paramètres intrinsèques, extrinsèques et des modèles 3D des objets. Le modèle mathématique utilisé pour générer ces rendus est le même que celui utilisé pour l'acquisition d'images, plus un changement de repère. De ce fait, les rendus obtenus se superposent parfaitement aux images

de la caméra. Ce qui permettra de mettre en œuvre notre application de réalité mixte et de générer plus de données annotées si besoin.

Reste à savoir comment obtenir les modèles 3D s'ils ne sont pas disponibles ? Comment mesurer les paramètres intrinsèques ? Et comment détecter en temps réel les paramètres extrinsèques ? Dans le prochain chapitre nous détaillerons ces trois problèmes.

Chapitre 3

Problématique

Introduction

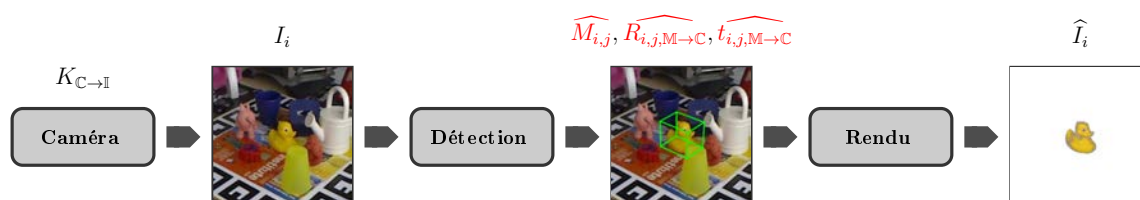


FIGURE 3.1 – Schéma illustrant le processus de fonctionnement d’une application de réalité mixte où apparaissent en rouge les paramètres qu’il faut pour que l’application puisse calculer ou estimer pour fonctionner.

Nous avons vu comment les images sont formées par une caméra ou par l’œil humain et comment l’exprimer mathématiquement par une suite de rotations, translations et projections de perspective. Nous avons aussi vu comment simuler ce processus pour obtenir des rendus 3D qui se superposent à ces images. Néanmoins, pour que notre application de réalité mixte fonctionne, il nous reste à résoudre un dernier problème :

Comment calculer les paramètres du système optique ?

Les paramètres que l’application doit connaître ou calculer sont illustrés en rouge dans la figure 3.1. Dans ce chapitre, nous rappelons pourquoi ces paramètres constituent un problème à résoudre. La première section est consacrée aux paramètres intrinsèques, la seconde aux paramètres extrinsèques et la dernière aux modèles 3D des objets.

3.1 Paramètre Intrinsèques

Les paramètres intrinsèques K sont essentiels. Ils caractérisent les propriétés optiques de la caméra utilisée et donc de la façon dont sont formées les images acquises par projection perspective. La plupart des caméras récentes, surtout celles pour la vision par ordinateurs comme les caméras Intel RealSense [®][64, 60], donnent directement les valeurs de ces paramètres ou permettent de les récupérer dans le logiciel ou leurs API de développement. D’autres, notamment à la focale ajustable, sauvegarde ces paramètres dans les métadonnées des images, ils sont donc directement accessibles.

Il arrive que ces paramètres ne soient pas disponibles ou pas communiqués. Par exemple sur des caméras plus anciennes, ou à bas coût. Ou encore lorsque l’on a accès à une base

d'images qui ne contient pas les informations des caméras qui ont acquis les images. Dans ce cas, il est possible de calculer ces paramètres à partir de plusieurs images présentant une mire de calibration, ou le même objet vu de plusieurs points de vue. Cette technique sera présentée dans la partie suivante consacrée à l'état de l'art.

Dans la suite de cette thèse nous considérerons que nous avons accès aux paramètres intrinsèques de la caméra, les jeux de données que nous utilisons communiquent la matrice K et nous avons constitué un jeu de données à l'aide d'une caméra Intel RealSense [64, 60] pour laquelle la matrice K est aussi connue.

3.2 Paramètre Extrinsèques

Les paramètres extrinsèques $R_{i,j,M \rightarrow C}$ et $t_{i,j,M \rightarrow C}$ doivent être ré-évalués chaque fois qu'un objet entre, bouge, ou quitte le champ de la caméra (la scène). Cela vaut aussi si la caméra bouge, ou si l'utilisateur bouge dans le cas des casques de réalité mixte. Autant dire que dans notre application, il faut les évaluer en temps réel.

Les jeux de données et la caméra utilisée pour constituer notre jeu de données fournissent des images couleur et des cartes de profondeur alignées. Les cartes de profondeur sont d'une grande aide pour résoudre le problème de la projection perspective et retrouver la distance à l'objet. Étant donné qu'elles sont calculées grâce à deux images infrarouges par la caméra, elles souffrent de problèmes inhérents à cette longueur d'onde : le verre et les surfaces lisses ou métalliques de couleur unie ne sont pas (bien) perçus. Ce qui est rédhibitoire dans les applications industrielles.

Pour inférer les paramètres extrinsèques, nous avons donc souhaité utiliser seulement les images couleurs fournies par la caméra. Plus les objets considérés seront nombreux et visuellement challengeants, plus cette tâche devient compliquée. C'est particulièrement le cas pour les applications destinées à l'industrie où les objets sont très nombreux et ont une apparence métallique, lisse ou transparente. Des objets peuvent aussi être identiques, modulo un facteur d'échelle (exemple : vis de différentes tailles). Il faut donc mettre en place un système de détection 3D robuste. Dans la prochaine partie consacrée à l'état de l'art, nous nous efforcerons d'identifier les méthodes qui peuvent répondre à ce problème.

3.3 Modèle 3D

Selon l'application, deux cas sont possibles : soit les modèles 3D des objets sont disponibles, soit ils ne le sont pas. Il y a même plus de chance que les modèles 3D ne soient pas disponibles. En effet, des produits ont été mis en service il y a des années, voire même des décennies, et continuent d'être exploités. Il n'y a aucun modèle 3D pour ces produits conçus avant l'ère du numérique. Des bâtiments et des usines ont été construits sur plans papiers. Ou bien encore des produits spécifiques comme des pièces sur mesure ou des petites séries peuvent ne pas avoir de modèle 3D. Il n'y a que dans le cadre de projets industriels récents que la conception assistée par ordinateur (CAO) s'est développée, et que les fabricants d'équipements d'origine (OEM) communiquent les modèles à leurs clients. La non-disponibilité de ces modèles 3D est un problème si l'application a besoin de les afficher. Par exemple, l'aide à la maintenance d'une machine ou le démontage d'une pièce est indiqué par l'affichage d'une animation du modèle 3D expliquant comment procéder. Comme nous allons le voir, les modèles 3D sont aussi très utiles pour pré-traiter les données d'entraînement et post-traiter la détection des paramètres extrinsèques.

Il est donc préférable qu'ils soient disponibles.

Fort heureusement, des techniques existent pour numériser des objets, voir même des bâtiments tout entier. Comme nous allons le voir dans la partie suivante consacrée à l'état de l'art, la communauté scientifique et industrielle s'intéresse à ce problème. Des avancées majeures dans ce domaine ont été faites, que ce soit sur le plan du matériel d'acquisition, ou sur la partie reconstruction logicielle. Dans la suite de cette thèse, nous considérons que nous avons accès au modèle 3D, soit parce qu'ils sont disponibles, soit parce qu'ils ont été calculés.

Puisque les modèles 3D sont connus, nous n'avons plus à détecter tous les points les composant ($M_{i,j,k} \forall k$). Néanmoins, il nous reste à identifier quels objets sont présents dans l'image I_i . On cherchera donc un indice $c_{i,j}$ permettant d'identifier à quelle catégorie d'objet appartient l'instance d'objet j .

Conclusion

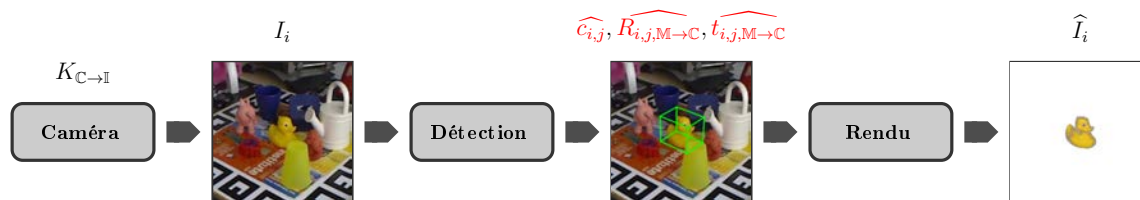


FIGURE 3.2 – Processus de fonctionnement d'une application de réalité augmentée ou apparaissent en rouge les paramètres dont nous traitons l'estimation dans cette thèse. Ici $c_{i,j}$ représente la catégorie de l'objet et non son modèle 3D. Et $R_{i,j}, t_{i,j}$ sont les paramètres extrinsèques.

Dans cette thèse nous considérerons donc que les paramètres intrinsèques de la caméra sont connus, et que les modèles 3D sont disponibles. Ou, que nous disposons de suffisamment de données pour les calculer, à l'aide de techniques existantes. Il nous reste donc à répondre au problème de la détection en temps réel des objets et des paramètres extrinsèques, comme illustré en figure 3.2. En effet, si les modèles 3D sont connus, nous n'avons pas besoin de les reconstruire en temps réel, nous avons seulement besoin de savoir lesquels sont présents à l'image. Les paramètres extrinsèques que nous allons inférer, eux en temps réel, nous permettront de savoir où ils sont situés en 3D dans l'environnement réel, et en 2D dans l'image. Il nous faut donc mettre en œuvre un système de vision par ordinateur capable de reconnaître les objets présents dans une image et d'être capable de mesurer leurs positions et orientations par rapport à la caméra.

Dans la partie suivante nous présenterons l'état de l'art des méthodes existantes permettant de répondre à nos problèmes

Deuxième partie

État de l'art

Introduction

Reconnaître et détecter la position et l'orientation d'un objet en trois dimensions est un problème que la communauté de chercheurs en vision par ordinateur cherche à solutionner depuis longtemps. Qui n'a jamais eu envie d'un ordinateur ou d'un programme informatique doté des mêmes capacités de perception que l'Homme, voir meilleures, et capable de nous assister ou même de nous décharger dans nos tâches quotidiennes. Les briques logicielles permettant de résoudre ce problème ont d'ailleurs été pensées au milieu du XX^{ème} siècle. La transformation de Hough utilisée dans de nombreuses méthodes a été publiée en 1959 [57, 30] et le perceptron [98], qui est l'ancêtre de l'apprentissage profond, en 1958.

Ce qui manquait aux chercheurs de l'époque était sans doute la puissance de calcul et l'espace mémoire nécessaires pour entraîner les modèles et traiter la masse de données. Ils ne disposaient pas non plus de capteurs capables d'acquérir des images en 2.5 ou même 3 dimensions et qui correspondent mieux au problème de la détection 3D.

Pour ces raisons, les méthodes de l'état l'art se sont longtemps limitées à l'utilisation des vecteurs de supports machines (SVMs) [121, 105] et aux forêts de classifications [52] couplés à des descripteurs "fait-main" tels les SIFT, SURF et HOG [75, 4, 37]. Nous verrons certaines de ces méthodes dans cette partie.

La Microsoft Kinect, dont la première version est sortie en 2010 sur Xbox et en en 2012 sur PC, a révolutionné la vision par ordinateur [131]. C'est la première caméra stéréo et vidéo couleur tout public. Elle était principalement destinée à l'industrie vidéo ludique. Le SDK open source a contribué à son succès, que ce soit pour des petits projets d'éducation, des projets individuels ou des projets de recherche. Aujourd'hui la troisième version de la Kinect, les Microsoft Azures, et de nombreuses caméras stéréoscopiques sont proposées par plusieurs fabricants, comme les Intel RealSense [64]. Comme leurs noms l'indiquent, ces caméras utilisent deux objectifs pour photographier deux images légèrement décalées, comme nos yeux le font. De ce fait, elles peuvent retrouver la distance à l'objet, mais elles produisent aussi deux fois plus de données. Dans le même temps, les GPUs toujours plus puissants destinés eux aussi à l'industrie vidéo ludique ont mis à disposition des chercheurs et des entreprises des bêtes de calculs. Elles embarquent aussi plus de mémoire vidéo, aujourd'hui jusqu'à 24 Go, ce qui permet de stocker les données au plus près des unités de calculs. L'algèbre linéaire hautement parallélisé sur ces machines a permis d'accélérer grandement le calcul des descripteurs, l'entraînement et l'inférence des méthodes.

Le Perceptron qui jusqu'alors connaissait un intérêt limité a énormément bénéficié de cette révolution. Il a évolué en modèles multi-neurones et multi-couches qui s'implémentent bien sur GPU du fait de leur nature parallélisée. La descente de gradient ré-implémentée sur GPU permet d'apprendre les poids du modèle très rapidement. Avec l'invention des couches de convolution, il a aussi été possible d'implémenter directement dans le modèle l'extraction des descripteurs. Puisque le Perceptron est entraîné sur les données, il tire aussi parti de leur plus grand nombre. Le Perceptron a alors très vite comblé l'écart avec les SVM [45], puis les a complètement surpassés comme nous allons le voir.

Chapitre 4

Méthodes de détection

Introduction

Dans ce chapitre nous présentons l'état de l'art des méthodes de détection 1D, 2D, puis 3D.

La première section est consacrée aux méthodes de détection 1D. Elles permettent de retrouver quels objets sont présents à l'image. L'application de réalités mixtes a besoin de cette information pour retrouver les modèles 3D des objets en base de données. Certaines méthodes de détection 2D et 3D sont spécialisées pour un seul objet. Elles auront donc aussi besoin de cette information pour fonctionner.

La deuxième section s'intéresse aux méthodes de détection 2D. Ces méthodes localisent les objets dans l'image. Elles sont suffisantes pour mettre en œuvre des applications de réalité augmentée simple sur des smart-glasses ou des affichages tête-haute par exemple. Comme nous allons le voir, elles s'inspirent des techniques de détection 1D, et ont inspiré les méthodes de détection 3D.

La troisième, et dernière section, présente les méthodes de détection 3D. Ce sont ces méthodes qui sont utilisées pour calculer les rotations et les translations des objets identifiés. Elles sont la clé du bon fonctionnement d'une application de réalité mixte complète telle que décrite dans la problématique.

4.1 Détection 1D

Introduction

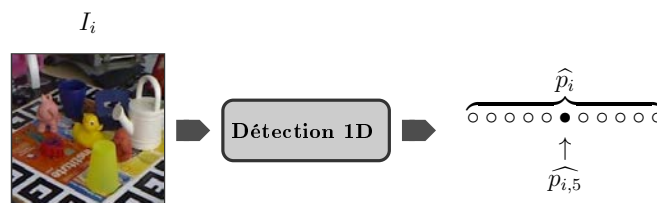


FIGURE 4.1 – Schéma illustrant un module de détection 1D. Les probabilités que chaque catégorie d'objets soient présents à l'image sont estimées. Ce qui permet de savoir quels objets sont présents dans l'image.

La détection 1D, aussi appelée classification d'image, a pour but de déterminer quels objets sont présents dans une image. Comme mentionné ci-dessus, cette information est très

utile pour notre application de réalité mixte. Elle permet de contextualiser les informations affichées à l'écran et d'adapter la suite des traitements à effectuer.

La classification d'images peut se résoudre de multiples façons. Si un grossiste en fruits cherche à trier ses fruits selon leurs niveaux de maturité, alors le seuillage de la couleur dominante peut suffire. La couleur est un descripteur utile mais elle est rarement suffisante. La texture est un autre descripteur qui permet de différencier plus d'objets. Pour la caractériser, des descripteurs locaux comme LBP ont été imaginés [89]. SIFT est un descripteur invariant à l'échelle et robuste à de nombreux facteurs de difficultés comme les changements de luminosités [75]. SURF est une version plus rapide à calculer et quasiment aussi robuste [4]. De façon à traiter des objets à la géométrie changeante, des modèles par parties basés sur des histogrammes de gradients orientés existent aussi [37]. Pour trancher entre ces descripteurs, les supports de vecteur machine [121, 105, 102] sont l'outil adapté. Ils permettent d'obtenir une probabilité d'appartenance à chaque catégorie d'objets considérée.

En 1998, Yann Le Cun a changé la donne avec le premier réseau de neurones convolutionnel entraîné bout-à-bout par descente de gradient [69, 110]. Les réseaux de neurones étaient jusqu'alors composés seulement de couches denses, i.e. complètement connexes. Ces couches contiennent un nombre de paramètres égal aux nombre de valeurs en entrées, multipliées par le nombre de valeurs en sorties. Si on cherchait à calculer un descripteur de 64 valeurs sur une image de 640×480 pixels, il faudrait déjà 19 660 800 paramètres. Sans compter les biais. Les couches de convolution mettent en commun les poids spatialement, ce qui permet de limiter le nombre de paramètres du réseau. De plus, les mêmes descripteurs seront calculés, peu importe l'emplacement dans l'image, ce qui rend le réseau invariant à la translation. Ainsi LeNet chargé de reconnaître les chiffres sur des chèques bancaires a obtenu 1 % d'erreur.

À l'époque de sa publication, il fallait trois jours pour entraîner LeNet sur du matériel spécialisé (FPGA). Depuis, la puissance de calcul de nos ordinateurs et en particulier de nos GPU a été décuplée. Aujourd'hui, il suffit de quelques secondes pour entraîner le même réseau sur un GPU. Comme nous allons le voir, les efforts menés par les chercheurs ont permis aux réseaux de neurones de dominer les challenges de vision par ordinateur et c'est particulièrement le cas pour la classification d'images.

Dans cette section, nous avons souhaité revenir sur les réseaux de neurones convolutionnels les plus connus, en mettant en avant des chiffres clés, comme le nombre de paramètres, l'empreinte mémoire. Nous avons sélectionné les réseaux qui ont fait progresser le domaine de la vision par ordinateur grâce à des innovations d'architecture.

4.1.1 AlexNet

Proposé en 2012, AlexNet [67] est le premier réseau de neurones convolutionnel ayant obtenu de bons résultats de classification sur des images complexes. Notamment sur ImageNet 2010 [25, 101] où il a obtenu 27 % d'erreur top-1¹. L'architecture d'AlexNet est présentée à la figure 4.2. Le réseau comprend un total de 62 378 344 paramètres, ce qui équivaut à 250 Mo. C'est un réseau d'une profondeur de huit couches. Cela paraît peu de nos jours, mais à l'époque de sa conception en 2012, il fallait deux cartes graphiques GTX 580 pour l'entraîner. Chaque carte dispose de 3 Go de mémoire vidéo, soit 6 Go au total. Les filtres des convolutions étaient répartis sur deux GPUs, offrant ainsi deux chemins de calculs parallèles, inter-connectés à certaines couches. Dans la figure 4.21 illustrant l'architecture bi-GPU, la colonne de gauche est implémentée sur le premier GPU et la colonne de droite sur le deuxième. Par exemple, la première couche de convolution du modèle mono-GPU comprend 96 filtres.

1. L'erreur top-1 mesure le pourcentage de résultat faux où un résultat est faux si la catégorie de l'objet n'est pas celle avec la plus forte probabilité à la sortie du réseau

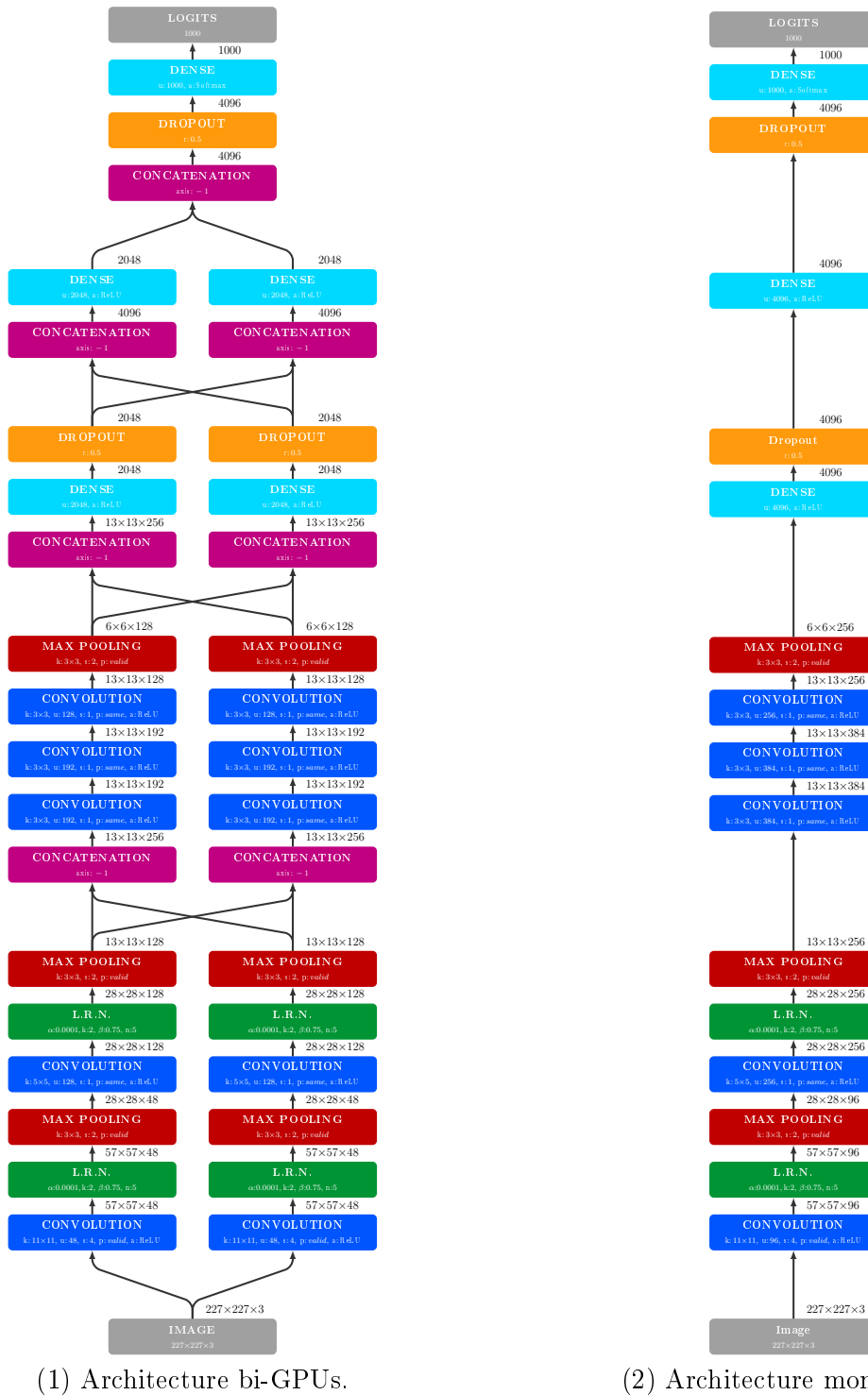


FIGURE 4.2 – Architecture du réseau de neurones AlexNet [67]. Dans la version bi-GPUs les concaténations ne réalisent aucune copie des tenseurs. Elles redirigent seulement la lecture des données sur le bon GPU selon l’index du canal lu. Par exemple, les concaténations précédant les convolutions de profondeur 3 redirigent la lecture des canaux 1 à 128 sur le GPU 1 et les canaux 129 à 256 sur le GPU 2. Dans la version bi-GPU chaque les couches sont connecté à moins d’entrées et de sorties, elles ont donc moins de paramètres. Le réseau mono GPU comprend donc plus de paramètres que le bi-GPU.

Dans le modèle bi-GPU, elle est divisée en deux couches, chacune de 48 filtres, réparties sur les deux GPUs. À l'époque, il n'était pas possible de rajouter plus de couches faute de mémoire et de capacités de calculs supplémentaires. Mais les auteurs avaient déjà conscience qu'ils disposaient de suffisamment de données d'entraînement avec ImageNet pour mettre en œuvre des réseaux beaucoup plus gros.

Le jeu d'entraînement ImageNet [25, 101] comprend 1.2 millions d'images. Lors de l'apprentissage 90 époques sont réalisées par paquets de 128 exemples, ce qui a pris trois jours. Comparé à LeNet, AlexNet comprend déjà deux couches de convolution supplémentaires et beaucoup plus de neurones dans les trois couches denses. Les images considérées sont aussi plus grandes que celles de MNIST, ce qui augmente aussi le nombre de paramètres de la première couche (227×227 contre 32×32 pixels). Certes, AlexNet dispose de plus d'images d'entraînement (1.2 millions contre 60 000) mais entraîner avec succès autant de neurones est difficile.

Pour faciliter l'entraînement, les auteurs ont utilisé plusieurs astuces :

- Les images d'entraînement sont modifiées de façon à avoir une valeur moyenne par pixel égale à 0. Pour ce faire, on soustrait à chaque pixel la valeur moyenne de ce pixel dans toutes les images.
- Des couches de normalisation de réponses locales sont ajoutées au réseau. Elles créent des "inhibitions latérales qui mettent en compétition les neurones adjacents". Concrètement, un neurone qui contribue plus à la décision de classification aura une plus forte activation. Il se verra donc attribuer un gradient avec plus d'amplitude. À l'inverse, les neurones adjacents qui ont été inhibés recevront moins de gradient. Ces couches contribuent à réduire l'erreur top-1 de LeNet de 1.4 %.
- Des couches de "dropout" sont ajoutées entre les dernières couches denses et vont aléatoirement éteindre certains neurones. Cela permet d'entraîner par alternance des combinaisons différentes de neurones. Elles limitent la sur-spécialisation du réseau dans les premières époques de l'entraînement ("over fitting").
- Les images sont augmentées avec une combinaison de recadrages aléatoires à différentes positions, de réflexions horizontales et de variations des principales couleurs. Cela permet d'accroître artificiellement le nombre d'images d'entraînement et de limiter encore la sur-spécialisation.

4.1.2 VGG-19

VGG est sans doute le réseau de neurones le plus utilisé en vision par ordinateur. Il a été proposé en 2015 par Karen Simonyan et Andrew Zisserman membres du "Visual Geometry Group". Il est destiné à la classification d'images du jeu de données ImageNet [25, 101] où il a obtenu 74.5% de précision top-1, i.e. 25.5% d'erreur top-1. C'est un réseau d'une profondeur de 19 couches : 16 couches de convolution suivies de 3 couches denses. Son architecture est présentée à la figure 4.3. Il comprend 143 667 240 paramètres, ce qui équivaut à 575 Mo. VGG est implémenté avec la bibliothèque Caffé rédigée en C++ et est aujourd'hui disponible dans toutes les bibliothèques d'apprentissage profond. Pour l'entraînement, quatre copies du réseau sont instanciées, une par GPU NVIDIA Titan Black. Ces cartes disposent de 2880 cœurs CUDA fonctionnant entre 889 et 980 MHz et de ont une capacité mémoire de 6 Go, soit 24 Go au total. Les paquets de 256 images d'entraînements sont répartis en quatre paquets de 64 images, un paquet par GPU. Le gradient est calculé indépendamment sur les quatre instances puis il est moyenné pour obtenir le gradient final. VGG est entraîné en effectuant 74 époques sur les 1.3 millions d'images d'entraînement d'ImageNet. Le taux d'apprentissage

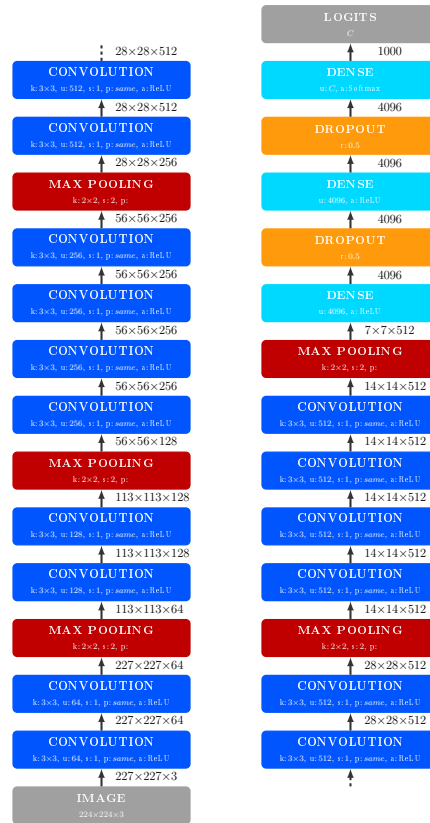


FIGURE 4.3 – Architecture du réseau de neurones VGG-19 [108].

initial de 0.01 est divisé par 10 chaque fois que la valeur de fonction de coût atteint un plateau. Au total, l'entraînement du réseau a pris trois semaines.

Les réseaux de neurones très profonds souffrent d'un problème de rétro-propagation du gradient [66]. Le gradient est calculé à partir de la couche de sortie puis, rétro-propagé vers les couches d'entrées. Plus on s'éloigne de la couche de sortie, plus la magnitude du gradient devient faible. Les couches hautes (proches de la sortie) sont donc entraînées plus rapidement que les couches basses (proches de l'entrée).

L'avantage de VGG est justement son bon rapport profondeur - disparition du gradient. Cette qualité tient à l'enchaînement de couches de convolution avec un noyau de 3×3 plutôt qu'une seule de couche de convolution avec un noyau plus grand. Cela permet d'augmenter la profondeur du réseau tout en utilisant moins des paramètres. Par exemple, trois couches avec des noyaux de 3×3 et une couche de 7×7 vont combiner le même nombre de pixels (elles ont le même champ réceptif), mais le premier cas utilise $3 \times 3 \times 3 = 27$ paramètres, alors qu'une seule couche de 7×7 en utilise 49. VGG peut donc être entraîné bout-à-bout (end-to-end) sans problème de disparition du gradient.

Pour mieux combattre les problèmes d'instabilité du gradient les auteurs ont aussi utilisés plusieurs astuces :

Le réseau est entraîné en plusieurs étapes. Il est d'abord constitué de 11 couches, puis 13, 16, et enfin 19. À chaque étape, les points du précédent sont utilisés pour initialiser le suivant.

Comme pour AlexNet, des augmentations d'images sont utilisées. Pour entraîner VGG les images sont aussi redimensionnées aléatoirement avant de les recadrer.

Une régularisation L_2 des poids du réseau est ajoutée à la fonction de coût avec une pondération de 0.0005. Cette régularisation permet de contraindre les valeurs prises par les poids du réseaux. En particulier, elle pénalise les grandes valeurs.

4.1.3 GoogLeNet

GoogLeNet [111, 112] est paru en même temps que VGG mais sa conception prend une autre direction. L'architecture de VGG est profonde, large et, d'un certain point de vu, dense du fait du nombre de paramètres. Les auteurs de GoogLeNet ont eux essayé d'utiliser le moins de paramètres possible tout en augmentant la profondeur du réseau. Il a une architecture un peu plus complexe mais qui fait une bien meilleure utilisation des couches de convolution. En particulier, il n'est pas séquentiel : il comporte différents "chemins" de différentes profondeurs. Ces différents chemins sont regroupés en modules appelés "Inception" qui sont présentés aux figures 4.41 et 4.42. L'idée est d'utiliser des couches de convolution différentes pour calculer des descripteurs à des échelles différentes (3×3 et 5×5). Ces chemins n'ont donc pas la même profondeur, mais surtout par le même champ réceptif. De plus, certaines convolutions sont précédées d'une convolution avec un noyau 1×1 mais avec moins de filtres. Cela permet de réduire le nombre de valeurs en entrée de la couche de convolution suivante, et donc le nombre de paramètres du réseau. On dit que la première convolution compresse l'information contenue dans le tenseur d'entrée. L'architecture complète est présentée à la figure 4.43. Le réseau cumule 13 607 784 de paramètres soit 54 Mo, c'est 10.6 fois moins que VGG ! Si on ne suit que les chemins les plus profonds, alors GoogLeNet est plus profond que VGG (22 couches contre 19). Du fait de cette profondeur accrue, le réseau est plus sujet à des instabilités de gradients. Pour pallier ce problème, deux sorties auxiliaires sont ajoutées pendant l'entraînement pour "réintroduire" du gradient plus bas dans le réseau. La même fonction de coût est calculée sur chacune des trois sorties et leurs valeurs sont ajoutées avec une pondération. La sortie principale est pondérée par un poids de 1 et les sorties auxiliaires par des poids de 0.3.

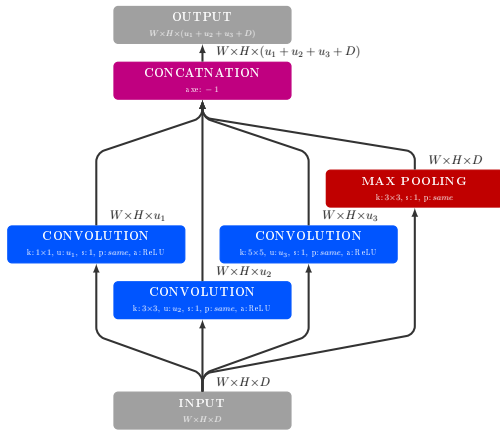
L'entraînement de GoogLeNet est effectué sur CPU. Sept instances sont implémentées avec à la bibliothèque DistBilief [23] qui permet de paralléliser l'entraînement. Les instances ont la même architecture (sauf une) mais l'échantillonnage des images n'est pas le même pour toutes. Contrairement à VGG, le gradient n'est pas combiné et les poids des sept instances sont indépendants. Lors de l'inférence chaque instance reçoit 144 images recadrées par image et produit 144 prédictions. Au total les sept instances produisent 1008 prédictions qui sont moyennées. GoogLeNet a terminé premier de la compétition ImageNet [25, 101] devant VGG grâce à son très bon score d'erreur top-5 de 6.67 %².

4.1.4 Res-Net

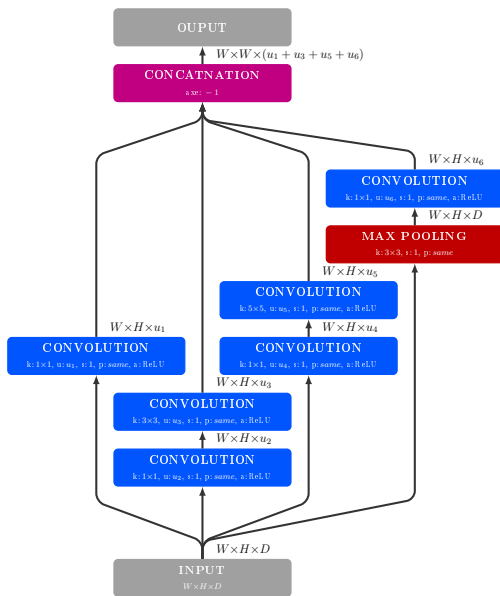
Dans un réseau de neurones convolutionnels, chaque couche apprend une fonction $f(x) = Wx + b$. Chaque couche transforme les données qu'elle prend en entrée en de nouvelles données en sortie. L'objectif est que les données à la sortie soient compressées (moins de valeurs) et plus caractéristiques du problème à résoudre. Un réseau de neurones convolutionnels va donc décomposer la fonction qui transforme une image en un descripteur utile par applications successives de couches. Nous avons vu avec VGG et GoogLeNet que plus le réseau est profond, plus il produit des descripteurs discriminatoires. De ce fait, la précision s'en trouve améliorée. Mais entraîner des réseaux de neurones convolutionnels très profonds est complexe du fait de l'instabilité et de la dissipation du gradient comme nous l'avons déjà mentionné [66].

Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun des équipes de Microsoft recherche ont eu en 2016 [48] l'intuition que la fonction apprise par une couche peut être reformulée comme $f(x) = g(x) + x$. Cette formulation réinjecte l'entrée de la couche (x) à

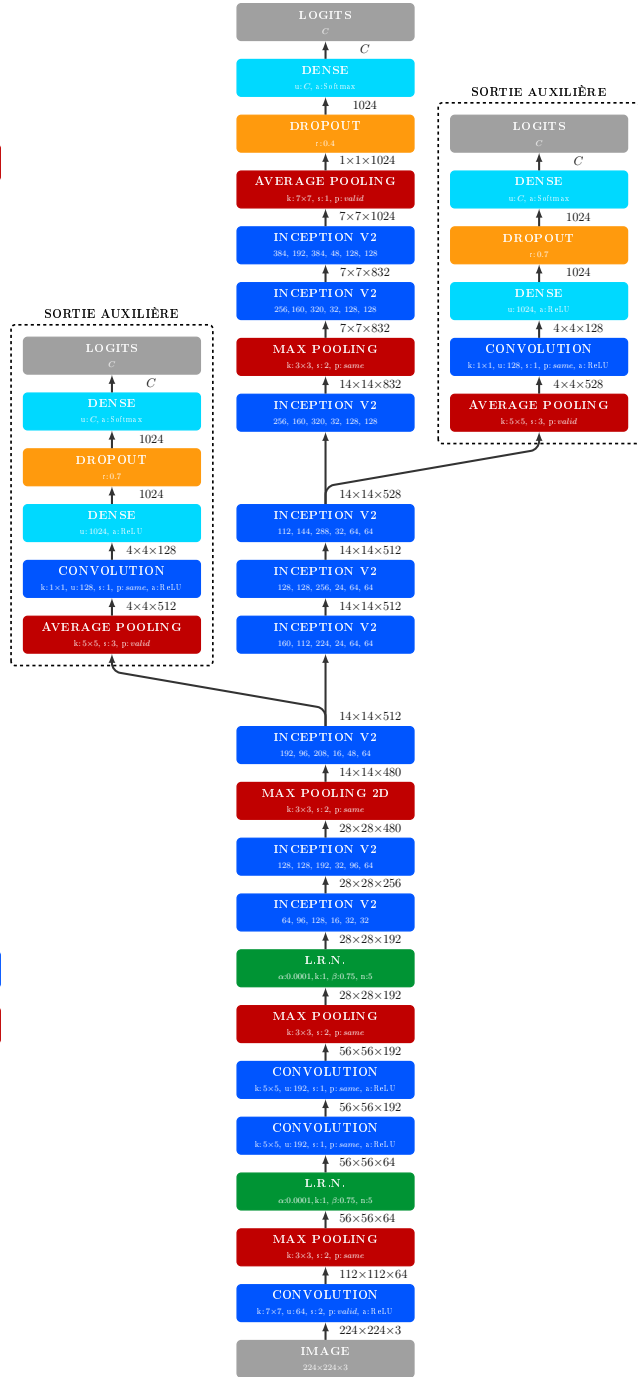
2. L'erreur top-5 mesure le pourcentage de résultats faux en déterminant qu'un résultat est faux si la catégorie de l'objet n'est pas une des 5 catégories comportant les plus fortes probabilités à la sortie du réseau



(1) Module Inception V1.

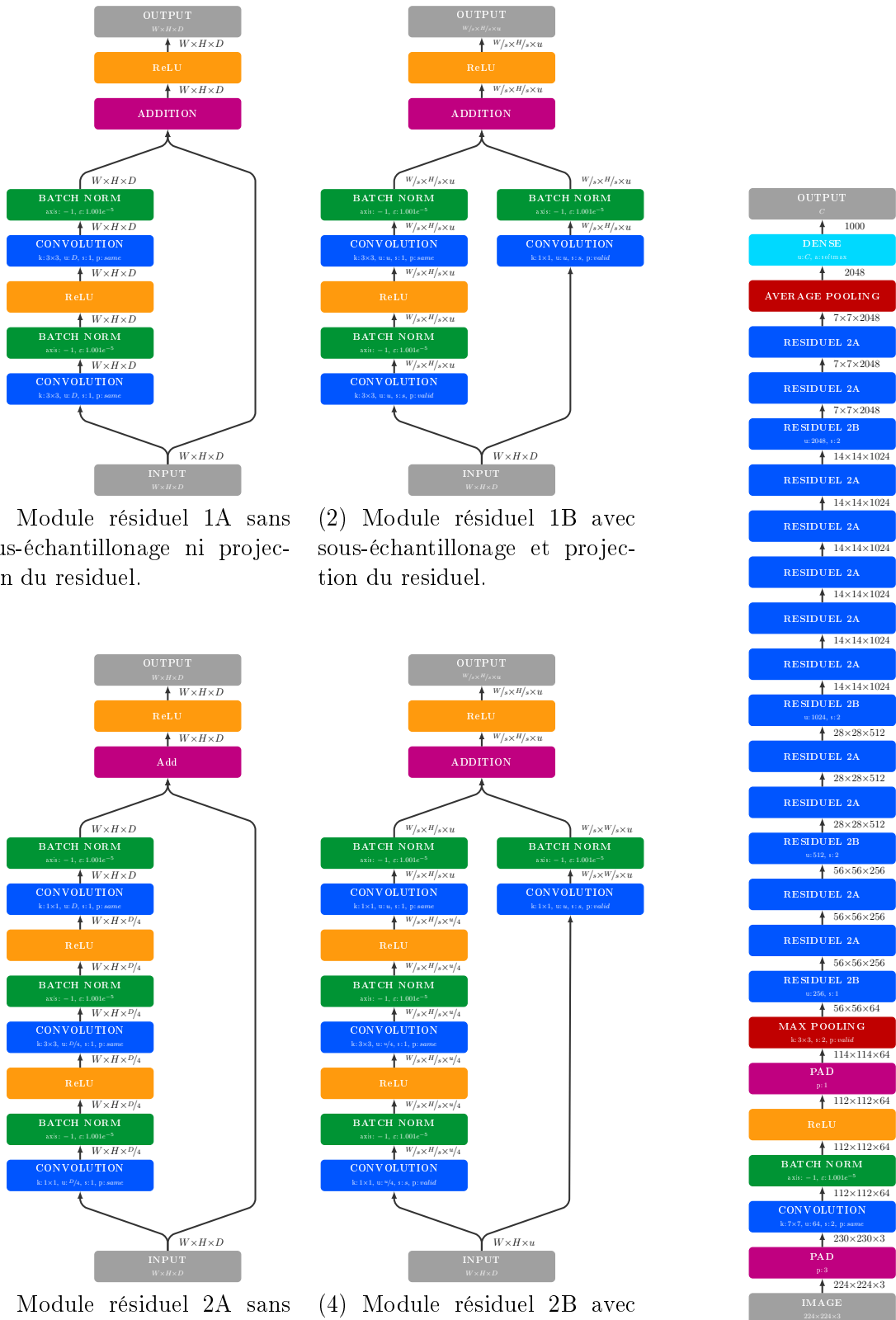


(2) Module Inception V2.



(3) Architecture de GoogLeNet.

FIGURE 4.4 – Architecture du réseau de neurones GoogLeNet [111].



(1) Module résiduel 1A sans sous-échantillonnage ni projection du résiduel.

(2) Module résiduel 1B avec sous-échantillonnage et projection du résiduel.

(3) Module résiduel 2A sans sous-échantillonnage ni projection du résiduel mais avec un gouleau d'étranglement.

(4) Module résiduel 2B avec sous-échantillonnage et projection du résiduel, et avec un gouleau d'étranglement.

(5) Architecture de ResNet50.

FIGURE 4.5 – Architecture des modules et du réseau ResNet [48].

sa sortie. Inversement, lors de la rétro-propagation du gradient, cela permet de réinjecter aussi le gradient en entrée de la couche. Les sous-fonctions $g(x)$ sont implémentées comme des modules. Ces modules peuvent aussi être considérés comme des sous-réseaux dans le réseau ("network in network"). Les expériences d'ablations menées par les auteurs ont montré qu'à même profondeur un réseau avec résiduel converge plus rapidement qu'un réseau sans. Dans le cas de ResNet [48], ces modules sont implémentés à l'aide de plusieurs couches de convolution : deux dans ResNet-18 et ResNet-34, trois pour ResNet-50, 101 et 152. L'architecture de ces modules est détaillée aux figures 4.53 et 4.54. Le réseau de neurones complet nommé ResNet-50 est présenté à la figure 4.55. Il comprend 60 419 944 paramètres (242 *Mo*) dont 60 268 520 sont entraînaibles. Les 151 424 paramètres fixes sont utilisés dans les couches de normalisation par paquets [61]. Ces couches assurent que les tenseurs ont une variance proche de zero. Ainsi, les activations dans le sens inférence, et le gradient dans le sens rétro-propagation, sont assurés d'être non-null. ResNet-152 a obtenu 21.43 % d'erreur top-1 et 5.71 % d'erreur top-5 sur le challenge ImageNet [25, 101].

4.1.5 Xception

Les convolutions réalisent des combinaisons linéaires entre les valeurs d'un tenseur spatialement (selon la largeur et la hauteur) et entre ses canaux (dans la profondeur). Hors, intuitivement, ces deux opérations sont différentes. Les dimensions spatiales sont liées à la position dans l'image alors que les canaux présentent des caractéristiques différentes. La convolution classique effectue donc des combinaisons entre plusieurs positions dans l'image et entre caractéristiques.

Laurent Sifre propose dès 2014 de décomposer l'opération de convolution en deux [120, 107]. La première effectue les combinaisons entre les canaux du tenseur et la seconde les combinaisons spatialement. Il a nommé cette opération la "convolution séparable" et a démontré que ce type de convolution améliore les performances d'un réseau comme AlexNet [67] mais améliore surtout la convergence lors de l'entraînement.

François Chollet a proposé en 2018 une architecture de réseau de neurones, inspiré des réseaux mobiles tel que GoogLeNet [111, 112] et MobileNet [58, 104] utilisant des couches de convolution séparables et des connexions résiduelles proposées pour l'architecture de ResNet [48]. Le réseau qu'il a créé est appelé Xception pour "extreme inception" [17]. L'architecture des modules Xception est illustrée à la figure 4.62 et l'architecture complète du réseau est détaillée à la figure 4.63. Le réseau comprend 22 910 480 paramètres (92 *Mo*) et à une profondeur de 37 couches sans passer par les connexions résiduelles. Les 22855952 paramètres des couches de convolution denses peuvent être appris et les 54 528 paramètres des couches de normalisation par paquets sont fixes comme pour ResNet.

Le réseau est implémenté à l'aide de la bibliothèque Keras [18]. Il est optimisé à l'aide de la descente de gradient stochastique classique [110] dont le moment est réglé à 0.9. Le taux d'apprentissage initial est de 0.045 et décroît de 6 % toutes les 2 époques. La régularisation L_2 des poids du réseau est aussi appliquée mais avec un facteur de 0.00001. Les résultats de classification d'images sont évalués sur le jeu de données ImageNet [25, 101]. Xception obtient 21 % d'erreur top-1 et 5.5 % d'erreur top-5 ce qui lui permet de se placer en tête devant GoogLeNet V3 (5.9 %), ResNet 152 (6.7 %) et VGG (9.9 %).

Conclusion Sur la Détection 1D

En l'espace de six ans, les réseaux de neurones ont évolué pour arriver à des scores de classification d'images quasi parfaits. Chaque avancée a été possible grâce à une meilleure

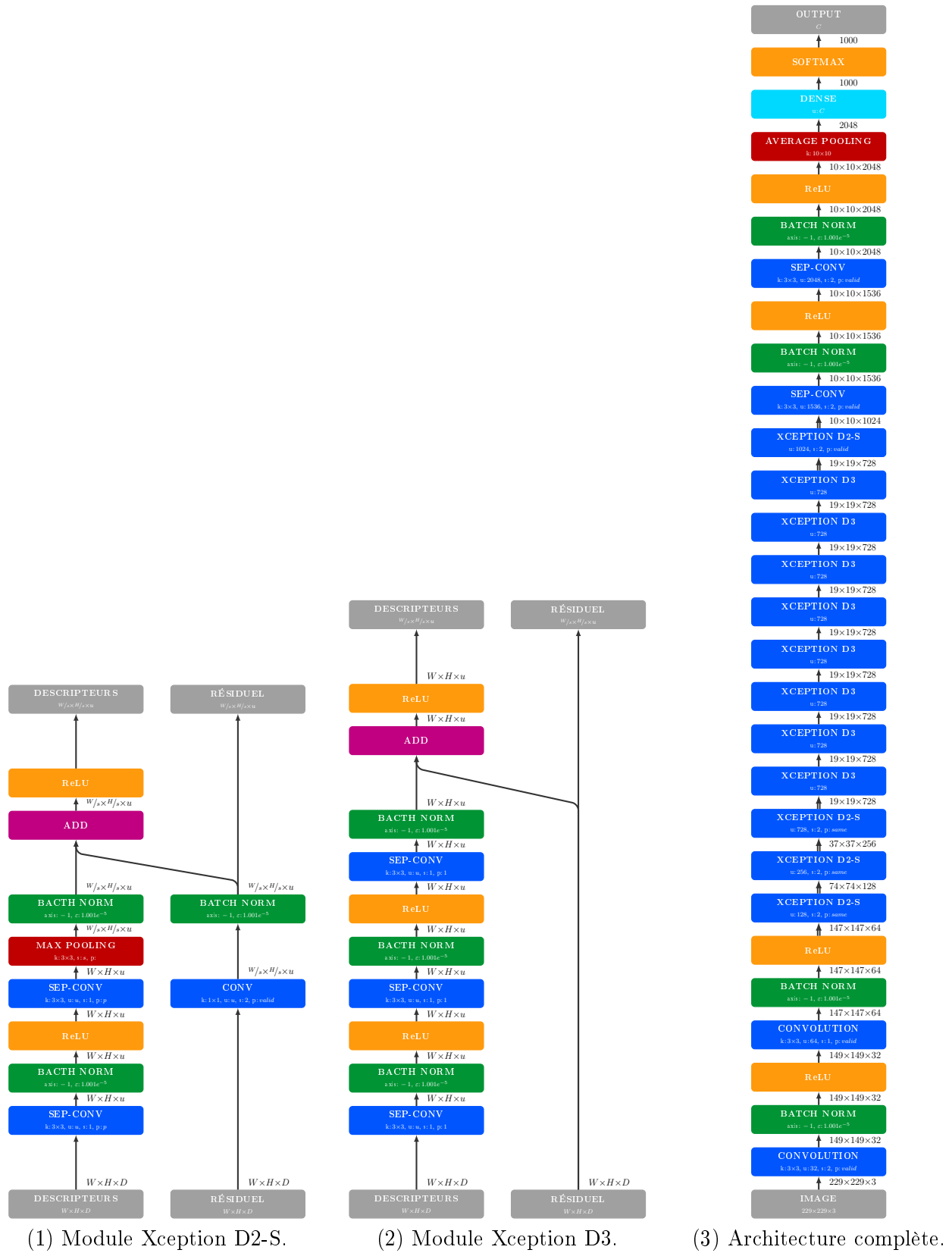


FIGURE 4.6 – Architecture d'Xception.

Méthode	Ref.	top-1	top-5
SIFT+FVs	[102]	50.9 %	73.8 %
AlexNet	[67]	63.3 %	84.6 %
GoogLeNet	[112]	71.6 %	94.5 %
ResNet-18	[48]	72.0 %	91.8 %
VGG-19	[108]	74.5 %	92.0 %
Inception v2	[61]	74.8 %	92.2 %
ResNet-50	[48]	77.2 %	93.3 %
ResNet-101	[48]	78.3 %	94.0 %
ResNet-152	[48]	78.6 %	94.3 %
Inception v3	[112]	78.9 %	94.5 %
Xception	[17]	79.0 %	94.5 %
EfficientNet-B7	[113]	84.4 %	97.1 %

TABLE 4.1 – Résultats de classification d’images des méthodes de l’art sur ImageNet [25, 101].

compréhension des architectures et de leurs mécanismes, comme c’est le cas pour les convolutions denses de VGG [108], la compression des tenseurs de GoogLeNet [111, 112], les convolutions séparables de Xception [120, 107, 17], et les connexions résiduelles de ResNet [48]. D’autres avancées ont aussi été réalisées pour optimiser les dimensions d’un réseau de neurones selon les ressources machines [113].

Les problèmes pour entraîner des architectures très profondes comme l’instabilité du gradient [66] sont aussi mieux compris aujourd’hui, ce qui a permis de mettre en place des stratégies de préparation des images comme la moyenne à zéro, des techniques de normalisation d’activations comme la normalisation de réponse local [67] ou plus tard la normalisation par paquets [61] et de meilleurs algorithmes de descente de gradient comme Adam [65].

Dans la prochaine section nous allons nous intéresser aux méthodes de détection 2D, i.e. de localisation d’objet sur une image.

4.2 Détection 2D

Introduction

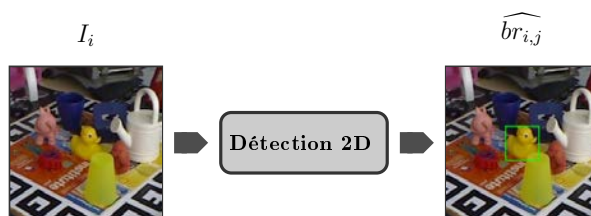


FIGURE 4.7 – Schéma illustrant un module de détection 2D. Les positions des objets à l’image sont estimées par des rectangles encadrants.

Nous avons vu qu’il était possible avec les réseaux de neurones convolutionnels de savoir quels objets sont présents dans une image. Pour pouvoir implémenter des applications de réalité mixte, nous avons aussi besoin de savoir où se situent ces objets par rapport à la caméra. Cette tâche est très complexe du fait de la perte de l’information de profondeur lors de la projection dans le plan image. Comme nous l’avons expliqué dans l’introduction et la problématique, les caméras de profondeur à notre disposition aujourd’hui fournissent

cette information qui nous fait défaut dans une image couleur. Connaissant les paramètres intrinsèques de la caméra, si nous pouvions détecter l'objet en 2 dimensions sur l'image, avec la profondeur donnée par la caméra, nous aurions assez d'informations pour calculer la position de l'objet en 3 dimensions. Même sans profondeur, l'algorithme "Perspective-N-Points" (PnP) permet de retrouver la rotation et la translation à partir de points détectés dans l'image dont on connaît les positions dans le repère objet. Détecter ces points en deux dimensions sur l'image est donc suffisant pour mettre en place des applications de réalités mixtes. Mais nous verrons ces méthodes dans la prochaine section. Détecter les objets sur l'image est aussi suffisant pour toute une famille d'applications de réalité augmentée qui n'affiche des informations que par superposition 2D. Comme par exemple les affichages têtes hautes dans les avions et les voitures.

Dans cette section nous présentons les méthodes de détection 2D. Ces méthodes ont évolué en s'inspirant des méthodes de classification d'images vues précédemment. Comme nous le verrons plus tard, elles sont elles aussi une source d'inspiration pour les méthodes de détections 3D. De ce fait, il est important de bien comprendre ces méthodes.

4.2.1 Approche Naïve

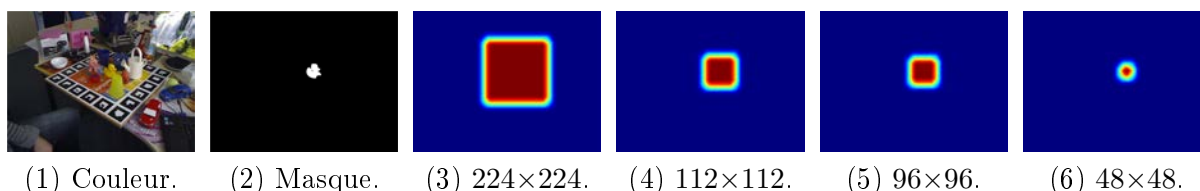


FIGURE 4.8 – Carte de chaleur de détection 2D naïve.

Nous avons à notre disposition des méthodes de classification d'images. Ces méthodes prennent en entrée une région carrée de l'image, et nous donnent en sorties la probabilité de chaque objet d'être présent dans cette région. Naïvement, nous pouvons balayer toute l'image de gauche à droite et de haut en bas avec notre méthode de classification d'images, ce qui produira une carte contenant pour chaque région évaluée les probabilités des objets. Cette carte peut être visualisée comme une carte de chaleur, par exemple comme illustré à la figure 4.8. On peut chercher dans la carte de chaque objet la zone la plus chaude, ce qui nous donnera la position de l'objet sur l'image.

Cette méthode fonctionne mais pose plusieurs problèmes. Tout d'abord, un très grand nombre de régions d'intérêt sont évaluées par la méthode de classification d'images. Pour une image de 640×480 pixels et un détecteur avec un champ réceptif de 224×224 le détecteur doit traiter 106 496 régions. Pour réduire la charge de calcul et gagner en temps, on peut augmenter le pas utilisé pour parcourir l'image. Par exemple, on peut décider d'évaluer des régions espacées de deux pixels, quatre pixels ou plus. En contrepartie, la résolution de la carte de chaleur sera diminuée et la localisation des objets sera donc moins précise.

Deuxièmement, avec un détecteur disposant d'un large champ réceptif, il n'est pas possible de localiser des objets apparaissant petit à l'image. Ils seront détectés, mais on ne pourra pas identifier de régions plus petite que celle du champs réceptif du détecteur. Par exemple, un détecteur avec un champ réceptif de 224×224 pixels n'est pas capable de localiser des objets plus petits que 224×224 . Pour résoudre ce problème, il est possible d'extraire des régions plus petites et de les redimensionner à la taille du détecteur. On pourra utiliser des pyramides [22] qui représentent l'image à plusieurs échelles. La figure 4.8 illustre les cartes de chaleur que l'on peut obtenir avec un détecteur dont le champs réceptif est de plus en

plus petit. Avec cette technique, la détection obtenue est plus précise, mais en contrepartie la méthode de classification d'images doit encore traiter plus de candidats.

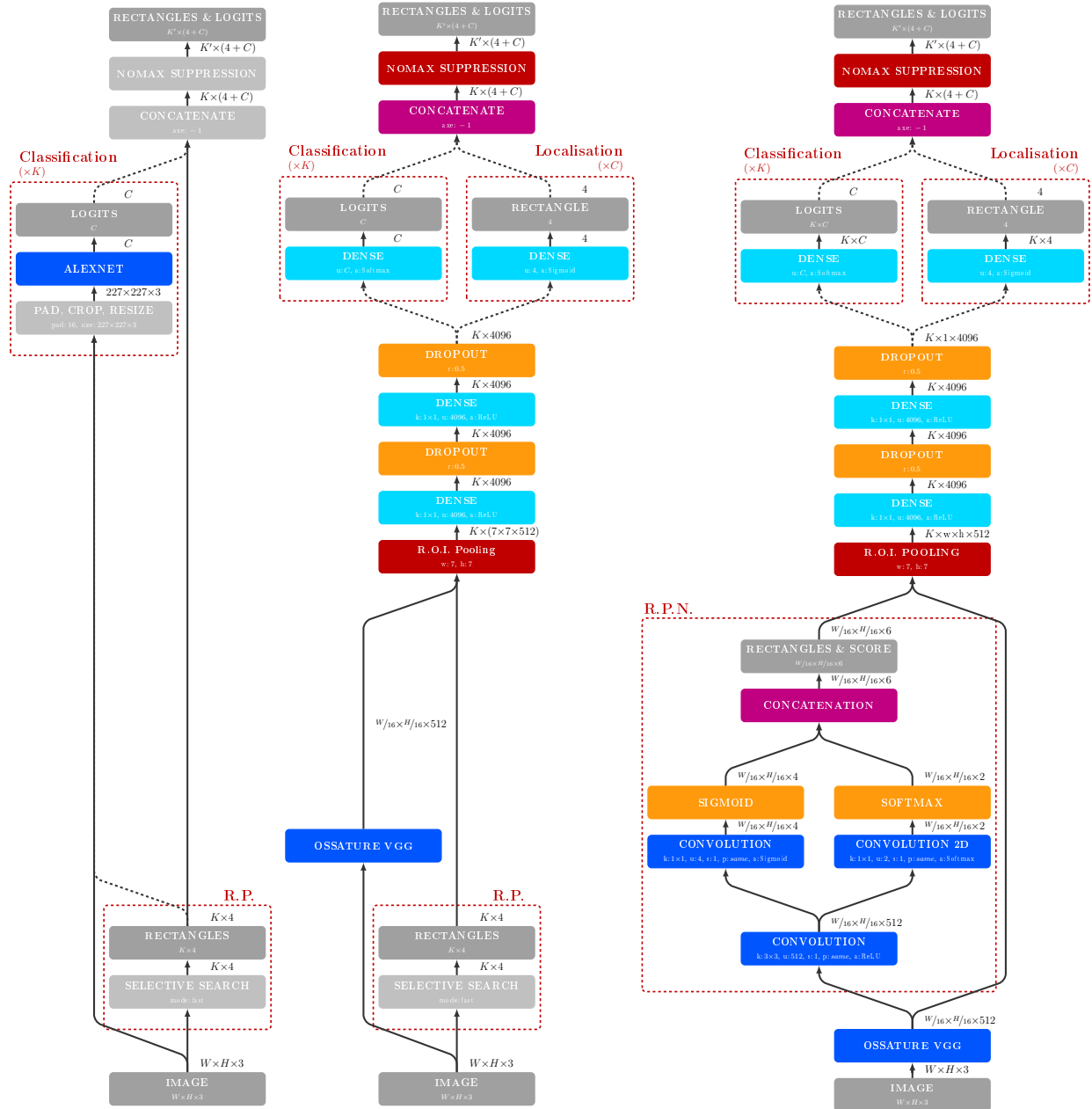
On comprend bien que la détection 2D naïve est limitée par les quantités de calcul liées au nombre de candidats qui doivent être considérés. Les réseaux de neurones peuvent donner des résultats pour plusieurs candidats simultanément grâce au traitement par paquet. Mais le nombre de candidats est si important dans le cas présent, que les limites mémoire seront très rapidement atteintes. Comme nous allons le voir dans le reste de cette section, la majeure partie des travaux présentés ici ont pour but de réduire le nombre de candidats considérés ou d'utiliser l'information de voisinage entre candidats.

4.2.2 R-CNN

En 2014 Ross Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik de UC Berkeley ont eu l'idée d'utiliser une combinaison de quatre méthodes pour résoudre le problème de la localisation d'objets [46]. Premièrement, une recherche sélective [118] est appliquée et propose un certain nombre de régions candidates. Cette étape permet de ne pas avoir à scanner toute l'image à la recherche des objets. Une segmentation de l'image est réalisée grâce à la technique des super-pixels [19, 35]. Ces super-pixels sont groupés hiérarchiquement de façon itérative. Configurée en mode rapide la recherche sélective propose de l'ordre de 1000 régions candidates. Deuxièmement, AlexNet [67] dont les paramètres ont été affinés sur les images de PASCAL VOC [33] est appliqué sur chaque proposition de région. Pour faire correspondre les régions proposées par la recherche sélective à la dimension du champ réceptif d'AlexNet, ces dernières sont élargies de 16 pixels dans toutes les directions pour inclure plus de contexte, puis elles sont redimensionnées à 227×227 pixels. Ces régions sont ensuite passées à AlexNet pour extraire 4096 descripteurs. En effet, ce n'est pas la sortie de la couche de classification qui est utilisée, mais celle de l'avant dernière couche de convolution. Celle-ci produit un vecteur de 4096 valeurs qui est hautement discriminatoire. Troisièmement, ce descripteur est utilisé par un SVM pour identifier l'objet présent dans les régions détectées [121]. Le modèle SVM est aussi entraîné sur Pascal VOC. Finalement, une suppression des non-maximum est utilisée pour élaguer les détections qui appartiennent à la même catégorie d'objet et partage une intersection sur l'union supérieure ou égale à 50 % [8]. Cette étape permet d'éliminer la plupart des faux positifs sans produire de faux négatifs. Optionnellement, les quatre coordonnées de chaque rectangle encadrant sont raffinées à l'aide de quatre fonctions linéaires. Les paramètres de ces fonctions sont appris par régression linéaire entre les coordonnées prédites, les valeurs de sortie de la cinquième et dernière couche de convolution d'AlexNet et les coordonnées étiquettes.

Au final R-CNN permet de prédire les coordonnées des rectangles encadrants sur le jeu de données PASCAL VOC avec une précision moyenne de 53.7%. Les prédictions sont calculées en 13 secondes pour une image sur un GPU et 53 secondes sur CPU. C'est très peu comparé aux 5 minutes nécessaires pour la méthode état de l'art à la même époque qui utilisait un modèle de parties déformables [24].

Mais R-CNN est compliqué à mettre en œuvre. Il faut entraîner AlexNet pour la classification d'image sur ImageNet. Puis l'affiner pour sur PASCAL VOC. Et enfin, entraîner le modèle SVM final sur PASCAL VOC lui aussi. Pour l'affinage d'AlexNet les auteurs ont utilisé un GPU Tesla K20 disposant de 2496 cœurs CUDA cadencés à 706 MHz et de 5 Go de mémoire vidéo. Ils ont réalisé 50 000 itérations de descentes de gradients (SGD) [110] avec des paquets de 128 exemples (32 positifs et 96 négatifs) avec un taux d'apprentissage de 0.001 ce qui a pris 14 heures.



(1) R-CNN [46]. (2) Fast R-CNN [44]. (3) Faster R-CNN [95].

FIGURE 4.9 – Architectures des réseaux R-CNN, Fast R-CNN et Faster R-CNN.

4.2.3 Fast R-CNN

Un an après avoir publié R-CNN [46], Ross Girshick a amélioré sa méthode de détection d'objets. Le principal défaut de R-CNN est son entraînement en plusieurs étapes. Déjà, à l'époque, il savait que le SVM pourrait être remplacé par une couche dense. De plus, les régions candidates sont extraites dans l'image et passées indépendamment au réseau de neurones pendant l'inférence et/ou l'entraînement. De ce fait, les ressources ne sont pas partagées pour effectuer les calculs. Pour répondre à ces deux problèmes, il a proposé en 2015 une amélioration de sa méthode nommée Fast R-CNN [44]. Premièrement, AlexNet est remplacé par l'ossature VGG pour augmenter les performances. L'ossature comprend les couches de convolution mais pas les couches denses. Deuxièmement, les régions d'intérêt produites par recherche sélective sont extraites dans le tenseur à la sortie de VGG et non dans l'image. Une nouvelle couche est implémentée pour effectuer le recadrage des régions et permettre la rétro-propagation du gradient. Troisièmement, le modèle SVM est remplacé par deux couches denses : une pour identifier à quelle catégorie d'objet appartient la région, et une autre qui régresse directement les coordonnées du rectangle encadrant. Grâce à ces deux améliorations, la méthode peut être entraînée bout-à-bout. Une seule passe d'inférence est requise pour traiter une image. Et le gradient est mis en commun entre les régions tirées d'une même image.

Lors de l'entraînement chaque paquet d'exemples est construit en extrayant 64 régions candidates sur deux images. Parmi ces 128 exemples, 25 % sont des exemples positifs dont l'IoU avec un objet est supérieur ou égale à 50 % et 75 % sont des exemples négatifs dont l'IoU est comprise entre 10% inclu et 50% exclu. Les exemples dont l'IoU est inférieur à 10% sont ignorés. L'entraînement et les tests sont effectués sur un GPU TESLA K40 disposant de 2880 cœurs CUDA cadencés à 745 MHz et disposant de 12Go de mémoire vidéo GDDR5. Fast R-CNN obtient une précision moyenne de 65.7 % sur le jeu de données PASCAL VOC 2012, c'est 12 % mieux que R-CNN.

4.2.4 Faster R-CNN

Fast R-CNN a permis de faire sauter presque tous les freins de R-CNN. Tous sauf la recherche sélective, qui, à l'époque, prend deux secondes par image. On est très loin des 30 images par seconde pour avoir un système temps réel. En 2016 Shaoqing Ren, Kaiming He, Ross Girshick et Jian Sun ont compris que la recherche sélective effectue des opérations similaires à l'ossature d'un CNN. Elle calcule des descripteurs bas-niveaux et les combine hiérarchiquement, niveaux après niveaux. Appliquer un CNN après une recherche sélective est donc aussi un gâchis du fait de la non mise en commun des calculs.

Pour la remplacer, ils ont proposé d'utiliser un nouveau concept basé sur un réseau complètement convolutionnel utilisé comme sous réseau. Premièrement, un réseau complètement convolutionnel remplace les couches denses par des convolutions avec des noyaux 1×1 et peut donc être appliqué comme une fenêtre glissante sur un tenseur pour produire une carte de prédictions. Deuxièmement, un réseau peut déléguer une tâche, comme la prédiction de rectangles encadrants, à un sous-réseau utilisé en interne. Les deux réseaux sont entraînés conjointement et travaillent ensemble pour produire le bon résultat. Le sous-réseau peut avoir une architecture adaptée au problème, ici complètement convolutionnelle. Autre contribution importante, le réseau n'essaie pas de prédire des rectangles de façon absolue (dans le repère de l'image, i.e. par rapport à ses coins), mais relativement au centre de l'image et relativement à des a priori de taille et d'échelle appelés "ancres". Cette expression facilite grandement l'entraînement de part une expression de la fonction de coût plus adaptée à l'entraînement des réseaux de neurones, et grâce à l'a priori des ancres qui donnent des objectifs

précis que le réseau n'a pas besoin d'intégrer.

Comme R-CNN et Fast R-CNN, Faster R-CNN utilise une colonne d'extraction de descripteurs, les auteurs ont choisi d'utiliser VGG. Vient ensuite le sous-réseau, qui a pour objectif de prédire des rectangles étant donné le tenseur produit par la dernière couche de convolution de la colonne d'extraction de descripteurs. Il a une architecture très simple, composée d'une couche de convolution avec un noyau 3×3 et deux couches de prédictions : une qui prédit les quatre coordonnées des rectangles encadrants et une qui prédit la probabilité que ce rectangle contienne un objet ou de l'arrière-plan. Ce sous réseau est appelé "réseau de proposition de régions" (RPN) et son architecture est détaillée à la figure 4.93. avec l'architecture complète de Faster R-CNN.

Faster R-CNN obtient 67.0% de précision moyenne sur le jeu de données PASCAL VOC 2012 [33]. Ce score est encore amélioré si plus de données sont disponibles pour l'entraînement : 73.2% avec PASCAL 2007 et 78.8% avec MS COCO [74]. Mais surtout, le réseau produit ces résultats à une fréquence de cinq images par seconde sur GPU ce qui permet enfin d'espérer une méthode temps réelle par exemple sur du matériel dédié ou avec une architecture simplifiée.

4.2.5 YOLO

La même année que Faster R-CNN, Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi ont proposé une méthode pour résoudre le problème de la détection 2D en temps réel [94]. Cette méthode, nommée YOLO pour "You Only Look Once", utilise une architecture mobile optimisée ainsi qu'une formulation élégante du problème de détection d'objets. Elle a, par la suite, été mise à jour [94, 92, 93, 7]. Nous allons ici présenter les deux premières versions en détail et la troisième brièvement.

V1

La première version de YOLO [94] utilise une architecture inspirée de GoogLeNet [111], à mi chemin avec Xception [17] mais beaucoup plus simple. Les modules Xceptions sont épurés au maximum pour ne garder qu'une combinaison de deux couches de convolution : une de compression et une d'expansion. L'architecture de ce module est illustrée à la figure 4.101. Comme pour Inception, la première couche dispose d'un noyau 1×1 qui recombine les canaux et compresse le tenseur d'entrée. La deuxième avec son noyau 3×3 recombine spatialement les descripteurs et produit deux fois plus de canaux que la première. L'architecture du module de réduction et de YOLO V1 est détaillée à la figure 4.104. Au total, le réseau comprend 271 703 550 paramètres soit 1087 Mo (1.09 Go). Le réseau est donc plus profond que VGG et comprend aussi plus de poids, mais grâce aux couches de compression, les tenseurs alloués entre les couches sont globalement moins volumineux.

La contribution majeure de YOLO réside dans sa formulation du problème de la détection d'objets. Au lieu de chercher à prédire un ensemble de rectangles sans corrélation avec l'image, YOLO découpe l'image en une grille. Chaque cellule est chargée de détecter un seul objet. Cette formulation est astucieuse car, premièrement, il est très improbable que deux objets soient visibles dans une même région de l'image surtout si on choisit une grille fine. Deuxièmement, la projection perspective déforme différemment les objets selon l'emplacement dans l'image. Il est donc logique d'avoir des détecteurs spécialisés pour chaque zone de l'image. Pour compléter ce découpage de l'image, la position des rectangles encadrants n'est plus prédite par rapport aux coins ou au centre de l'image mais par rapport au centre de chaque zone. La taille des rectangles est, elle, toujours relative à la taille de l'image.

Méthode	Ref.	mAP
R-CNN	[46]	58.5 %
YOLO	[94]	63.4 %
Fast R-CNN	[44]	70.0 %
Faster R-CNN	[95]	73.2 %
Faster R-CNN+ResNet-101	[48]	76.4 %
YOLO V2	[92]	78.6 %

TABLE 4.2 – Résultats de détection d’objets obtenus par les méthodes de l’état de l’art sur Pascal VOC 2007 [33].

YOLO obtient 57.9% de précision moyenne mais fonctionne à une fréquence de 45 images par seconde. Sa précision moyenne est donc $1.3\times$ inférieure à Faster R-CNN mais il est $5\times$ plus rapide. L’analyse menée par les auteurs montre que YOLO a du mal à localiser finement les objets comparé à Fast R-CNN, mais qu’il commet moins de faux positifs (arrière-plan identifié comme objet d’intérêt).

V2

Faster R-CNN a démontré qu’une méthode robuste pour augmenter la précision de localisation est d’utiliser des a priori sur la forme des rectangles encadrants : les ancres. Cette technique stabilise l’entraînement et augmente de plus de 12% la précision moyenne. YOLO V2, proposé à la fin de l’année 2016, adapte cette technique au détecteur temps réel YOLO V1.

Tout d’abord l’architecture du réseau est mise à jour. La normalisation par paquet est systématiquement appliquée après chaque convolution. Les convolutions ont, soit un noyau de dimensions 3×3 y compris à l’entrée du réseau, soit 1×1 pour les couches de réductions. Les couches denses sont remplacées par des couches de convolution de façon à rendre le réseau complètement convolutionnel. Une connexion entre plusieurs niveaux du réseau est aussi ajoutée pour apporter des descripteurs de grande échelle aux couches hautes. L’architecture complète est détaillée à la figure 4.105. Dans un deuxième temps, la formulation des étiquettes et donc des rectangles bruts prédits par YOLO est modifiée pour introduire les ancres. La principale différence avec Faster R-CNN est que les ancres ne sont pas choisies manuellement. Elles sont choisies par groupement K-Moyenne sur les rectangles encadrants de l’ensemble d’images de validation. De ce fait, elles correspondent mieux au jeu de données considéré et l’entraînement est d’autant plus stabilisé et accéléré.

V3

La version 3 approfondit l’architecture de YOLO, ajoute des connexions résiduelles inspirées de ResNet et produit des résultats à de multiples échelles. Pour ce faire, les couches de prédictions sont instanciées à trois profondeurs différentes pour produire des résultats à trois échelles. Les ancres sont aussi spécifiques à chaque échelle. Dans les couches basses, les ancres décrivent des objets de grande aire, alors qu’au sommet du réseau, les ancres décrivent des objets avec une petite aire.

Conclusion Sur la Détection 2D

La détection d’objets a évolué rapidement à l’aide des réseaux de neurones. Ils ont d’abord remplacé tous les blocs du processus de détection, à commencer par l’extraction de descrip-

teurs, puis la classification et enfin la détection des régions elles-mêmes. Mais c'est seulement quand les CNNs sont devenus complètement convolutionnels qu'ils ont pu apporter une solution complète à ce problème. Les performances atteintes et les temps de calculs nécessaires ont alors été sans précédent.

Dans la prochaine section nous allons nous intéresser aux méthodes de détection 3D, i.e. l'estimation de la pose des objets.

4.3 Détection 3D

Introduction

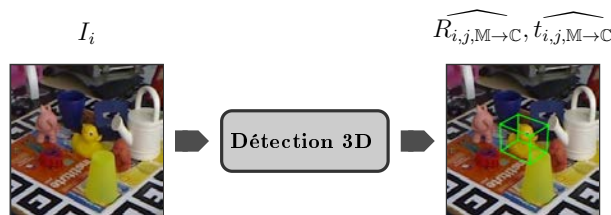


FIGURE 4.11 – Schéma illustrant un module de détection 3D. Les positions $t_{i,j,M \rightarrow C}$ et les orientations $R_{i,j,M \rightarrow C}$ des objets sont estimées.

La détection 3D, ou l'estimation des 6 degrés de liberté d'un objet, est le problème au cœur de cette thèse. Ces six paramètres, en plus de la catégorie de l'objet, permettent à une application de réalité mixte de savoir où se trouve l'objet dans son référentiel, et lui permettent d'augmenter visuellement ce dernier de façon immersive et réaliste pour l'utilisateur.

Historiquement, comme pour la détection 1D et 2D, des descripteurs faits mains étaient utilisés pour résoudre ce problème. Nous commencerons par présenter certaines de ces méthodes, car de nombreux jeux de données encore utilisés aujourd'hui sont conçus pour leurs entraînements. Puis nous présenterons les techniques utilisant des réseaux de neurones qui se sont imposées de nos jours.

4.3.1 Patrons de Pose

Pour détecter un objet en deux dimensions, la méthode de choix dans les années 2000 était le modèle de patrons déformables [34]. Les parties d'objets sont identifiées et localisées grâce à des descripteurs pré-calculés sur des images d'entraînement. Les descripteurs sont, par exemple, des histogrammes de gradients orientés [5, 37] qui sont robustes à de nombreux facteurs de difficultés comme les changements de luminosité, et permettent de traiter des objets sans textures discriminatoires. Cette technique a été étendue pour la détection 3D. En effet, une vue d'un certain angle et à une certaine distance, peut être considérée comme une "partie". La distance à l'objet étant directement proportionnelle à sa taille dans l'image, il est possible de la modéliser comme un paramètre de DPM. SVM et DPM permettent alors de savoir quelle "partie" est vue, et donc de retrouver l'angle et la distance à l'objet. Les modèles 3D des objets sont connus. Si en plus des cartes de profondeur sont accessibles, alors il est aussi possible d'intégrer la géométrie de l'objet dans les calculs des descripteurs [49, 51, 50, 13, 114, 55]

Les jeux de données LINEMOD et T-LESS ont été conçus pour entraîner et tester ces méthodes. Nous les présenterons dans le prochain chapitre.

4.3.2 Forêt de Hough

Une autre technique consiste à entraîner une forêt de décision [28]. Chaque arbre traite un pixel et prédit les coordonnées qui lui sont associées dans le repère objet [9, 10]. La transformation permettant de passer du repère objet au repère de la caméra peut ensuite être calculée. Cette dernière correspond aux 6 paramètres recherchés.

La forêt est entraînée à l'aide de rendus des modèles 3D des objets et testée sur les images de LINEMOD. Les auteurs ont aussi proposé une extension à LINEMOD contenant plus d'occlusions appelée OCCLUDED que nous présentons au prochain chapitre.

4.3.3 Réseaux estimant directement la pose

PoseNet et PoseNet++ Les réseaux de neurones convolutionnels sont de puissants outils d'apprentissages supervisés. Comme nous l'avons vu, ils fournissent une solution directe à de nombreux problèmes. Alex Kendall, Matthew Grimes et Roberto Cipolla de l'Université de Cambridge ont donc appliqué GoogLeNet [111] pour localiser une caméra dans un environnement connu [63]. Pour modéliser la pose de la caméra, GoogLeNet prédit sept valeurs réelles : un vecteur de translation en mètres et un quaternion pour la rotation (qui peut être convertie en angles d'Euler par la suite). Il ont obtenu une précision de 2 m et 3° pour des scènes en extérieur et 0.5 m et 5° en intérieur. Cette méthode a été étendue par l'utilisation de VGG comme architecture [130] au lieu de GoogLeNet, ce qui permet d'apprendre à prédire directement les angles d'Euler.

Le problème de localisation de caméra produit un résultat similaire à celui du problème d'estimation des six degrés de liberté des objets. La méthode régresse l'inverse des paramètres recherchés en détection 3D.

PoseCNN Inspirés par les résultats que les réseaux de neurones convolutionnels ont obtenu en segmentation d'images et en localisation d'objets, Yu Xiang, Tanner Schmidt, Venkatraman Narayanan et Dieter Fox ont proposé PoseCNN [126]. Cette méthode régresse aussi directement les paramètres de pose mais procède par étapes. Premièrement, une carte de segmentation dense est prédite et permet de dire à quel objet appartient chaque pixel. En parallèle, la position du centre des objets et leur distance à la caméra sont aussi prédites grâce à des cartes denses. Les coordonnées du centre et la distance à la caméra permettent de calculer le vecteur de translation de l'objet. Deuxièmement, les régions d'intérêt de chaque objet sont extraites grâce aux positions précédemment calculées par deux couches ROI Pooling comme dans Fast RCNN [44]. Troisièmement, chaque région est traitée pour régresser la rotation sous la forme d'un quaternion comme pour PoseNet. Pour finir, la pose est affinée par ICP [6].

PoseCNN est entraîné avec des images réelles. Le réseau obtient 24.9 % de précision ADD-3D sur LINEMOD OCCLUDED lorsque seul les images couleurs sont disponibles et sans ICP. Si la carte de profondeur est utilisée comme un canal supplémentaire en entrée et que l'ICP est appliqué, alors la précision moyenne est de 78.0 %. Sur les images couleur YCB-VIDÉO et sans ICP, PoseCNN obtient 53.7 % de précision ADD-3D et 75.9 % de précision ADD-3D-S. Avec les cartes de profondeur, la précision ADD-3D atteint 64.6 % et la ADD-3D-S atteint elle 83.7 %.

Les résultats de PoseCNN montrent que les cartes de profondeur sont essentielles pour prédire directement les paramètres extrinsèques. Elles seules sont capables de fournir au réseau une donnée permettant de calculer la distance à l'objet. Ils montrent aussi que les jeux de données LINEMOD et LINEMOD OCCLUDED, pour lesquels le réseau est 28.8 % moins performant, ne sont pas adaptés pour entraîner ou tester un réseau de neurones. Au

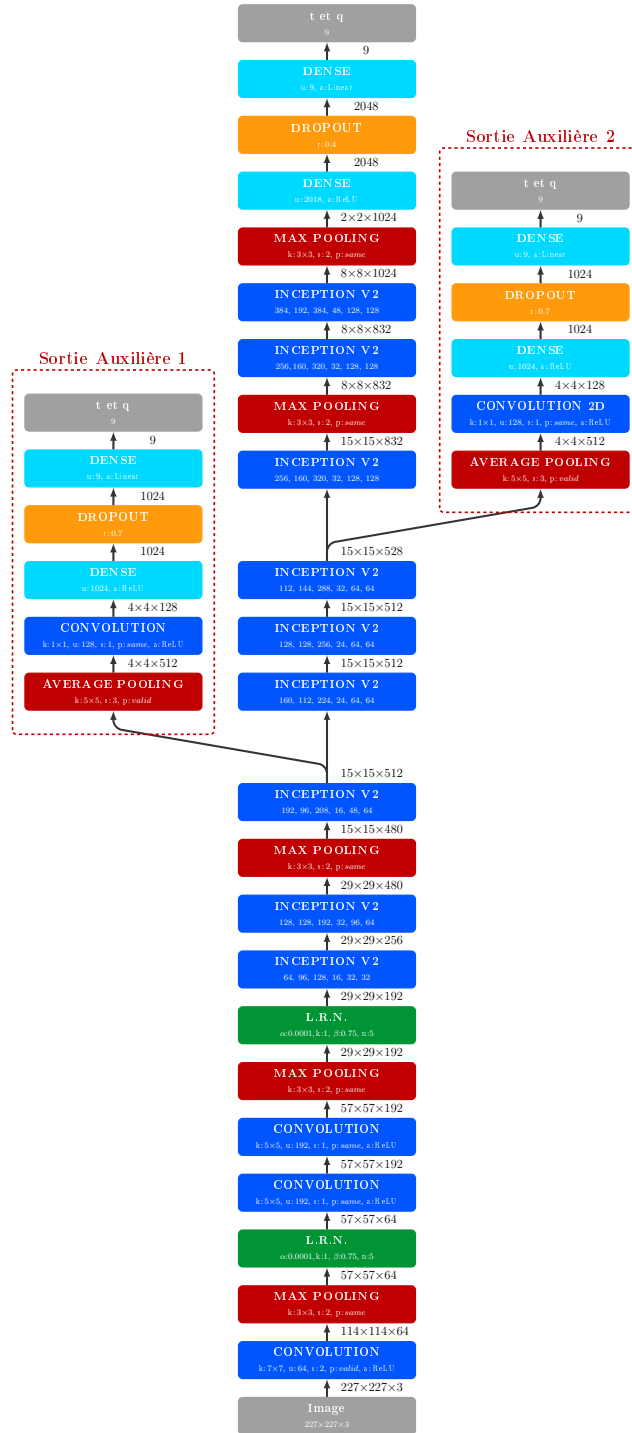


FIGURE 4.12 – Architecture du réseau de neurones PoseNet [63] basée sur celle de GoogLeNet [111].

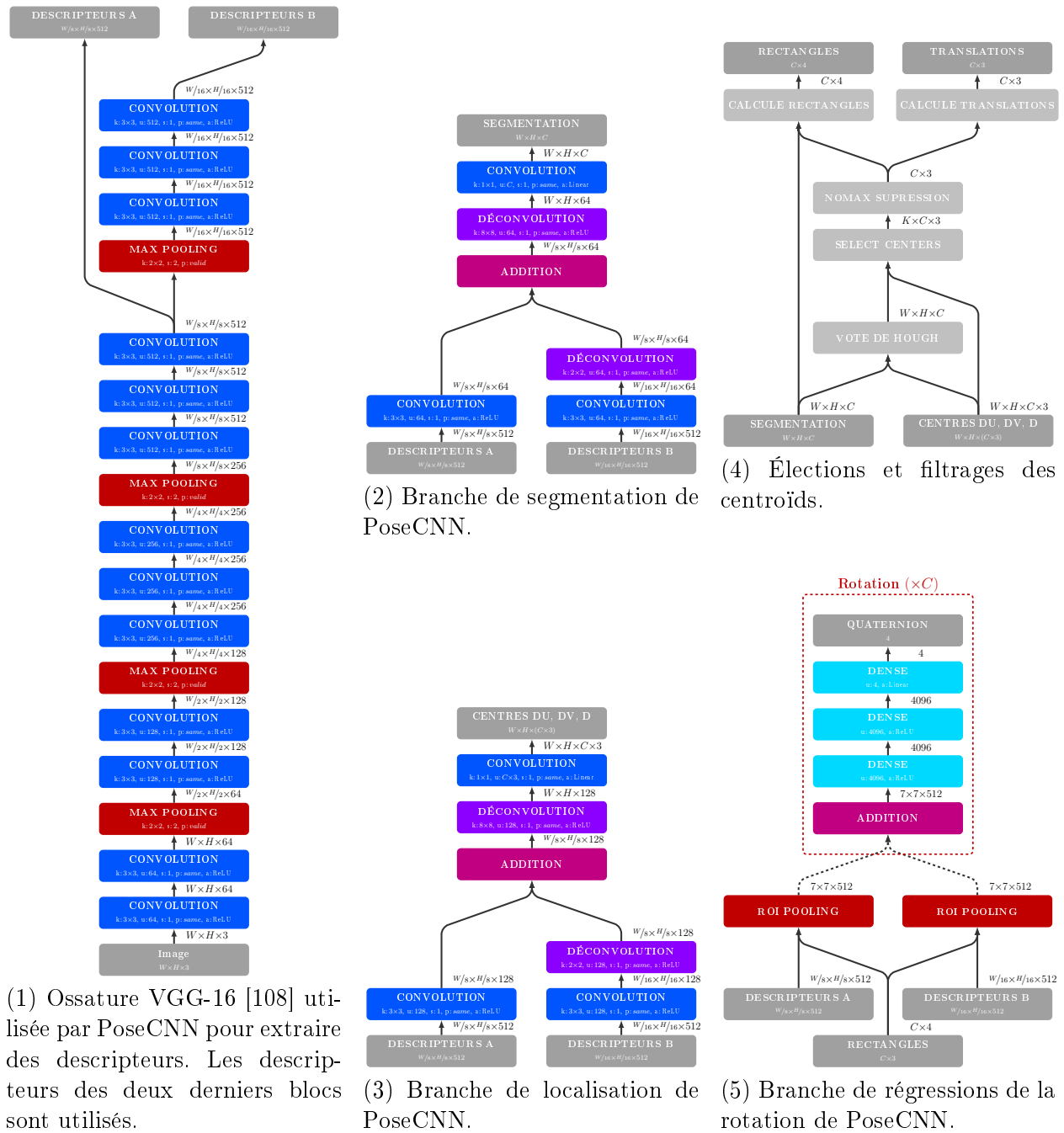


FIGURE 4.13 – Architecture de PoseCNN [126].



(1) Architecture de BB8 [91]. Le réseau prédit 8 points 2D pour chacun des C objets considérés. Les poids de la dernière couche dense sont donc indépendants pour chaque catégorie d'objets.

(2) Architecture de YOLO 3D [115]. Le réseau prédit une grille de $S \times S$ cellules contenant chacune des prédictions pour 9 points 2D, une valeur de confiance et C probabilités pour C catégories.

FIGURE 4.14 – Architectures de BB8 et Single Shot 6D Pose Prediction.

prochain chapitre nous apporterons une analyse plus approfondie des jeux de données et tenterons d'expliquer pourquoi ce problème se pose.

4.3.4 Correspondances de Points 2D-3D

Perspective-n-Point désigne une famille d'algorithmes qui ont pour but de trouver la matrice de paramètres extrinsèques étant donnée la matrice de paramètres intrinsèques, un ensemble de points définis dans le repère 3D de l'objet, et leurs projections dans le plan image. Plusieurs solutions à ce problème ont été apportées dans la littérature et permettent notamment de résoudre ce problème avec une complexité de $O(n)$ [76, 71]. Ces méthodes fonctionnent bien, même avec peu de points : quatre ou cinq. Elles ont déjà largement été utilisées avec des méthodes utilisant des descripteurs "fait main" et des SVMs. Nous avons vu à la section précédente que les réseaux de neurones convolutionnels sont capables de localiser en 2D dans le plan image des objets ou des parties d'objets. Ils surpassent aujourd'hui les méthodes classiques dans ce domaine. Il est donc possible de tirer parti des réseaux de neurones pour prédire la position sur le plan image de points dont la position dans le repère objet est connue, puis d'utiliser PnP qui fournit une solution directe et efficace au calcul des paramètres extrinsèques. Cette idée a conduit à de nombreuses méthodes de détection 3D très efficaces.

BB8

Mahdi Rad et Vincent Lepetit ont proposé en 2017 d'utiliser VGG pour prédire les coordonnées des huit coins de la boîte encadrante d'un objet, et d'utiliser PnP pour retrouver la pose [91]. Il est aussi possible d'utiliser plus de points pour chaque coin, ou partie d'objet [21]. L'architecture modifiée de VGG utilisée par BB8 pour prédire les huit coins de la boîte encadrante est présentée à la figure 4.141. Dans BB8, VGG est utilisé pour prédire les coordonnées en deux dimensions des points. Un autre VGG est, lui, chargé de segmenter l'image pour savoir quels objets sont vus.

BB8 est entraîné avec des images couleur. Sur LINEMOD, le réseau obtient 83.9 % de précision avec la mesure ADD-2D. Avec un VGG supplémentaire utilisé pour affiner la prédiction de pose, BB8 obtient 89.3 % de précision ADD-2D, 62.7 % de précision ADD-3D, et 69.0 % de précision à 5 cm et 5°.

YOLO 6D

La deuxième approche consiste à tirer partie des architectures dédiées à la détection d'objets comme YOLO V2 [94, 92]. Ces dernières permettent de traiter des images en une seule passe grâce à leurs architectures complètement convolutionnelles. C'est ce qu'ont proposé Bugra Tekin, Sudipta N. Sinha et Pascal Fua en 2018 [115]. Pour la détection 2D, ce réseau prédit les coordonnées d'un rectangle encadrant ce qui revient à deux points. Pour la détection 3D, on a besoin d'au minimum quatre points pour appliquer PnP. Plus de points permettront une meilleure résistance aux occlusions partielles mais impacteront la vitesse d'inférence de PnP. Les auteurs utilisent neuf points dans YOLO 3D : les huit coins de la boîte encadrante et le centre de gravité de l'objet. L'architecture de YOLO 3D est présentée à la figure 4.142.

YOLO 6D est entraîné avec des images réelles. Sur LINEMOD la méthode obtient 90.37% de précision ADD-2D et 55.95 % de précision ADD-3D.

4.3.5 Correspondances de Points 2D-3D Denses

R-CNN, YOLO, BB8 et SS6DPP ont montré que les réseaux de neurones convolutionnels sont capables de prédire les positions dans l'image de points en rapport avec l'apparence des objets ou des parties d'objets qui sont visibles dans celle-ci. Ces réseaux modélisent la position dans l'image en fonction des pixels des objets visibles dans l'image. La manière de combiner l'information des parties d'objets est laissée libre au réseau lors de son entraînement. Or, intuitivement, une approche statistique comme DPM est suffisante voir même meilleure dans le cas où des occlusions sont présentes. Externaliser cette étape permettrait donc de soulager le CNN et de consolider la fusion d'informations.

Pour ce qui est des architectures de CNN, les réseaux complètement convolutionnels permettent de prédire une information spatialisée qui met en rapport des zones de l'image. Ils sont déjà utilisés pour la détection d'objets et la prédiction de points utilisés pour calculer les poses d'objets. Les encodeurs-décodeurs comme UNet [97] sont capables de prédire des informations denses au niveau de chaque pixel. Ils sont utilisés par exemple pour la segmentation d'images ou la segmentation d'instances d'objets.

Il serait donc avantageux d'utiliser un encodeur-décodeur complètement convolutionnel pour prédire des informations de localisation de parties d'objets au niveau de chaque pixel. Ces nombreuses prédictions peuvent ensuite être traitées par une approche statistique de façon à obtenir la pose de l'objet globale.

Estimation Profonde de la Pose d'Objets

L'estimation profonde de la pose d'objets ("*Deep Object Pose Prediction*", i.e. DOPE) est une méthode qui a été proposée par Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox et Stan Birchfield des équipes de NVIDIA en 2018 [117]. Ils utilisent une version complètement convolutionnelle de VGG. Au lieu de prédire directement les coordonnées des points, leur réseau prédit neuf cartes de chaleur et huit cartes de directions. Les points chauds dans les neuf cartes de chaleur permettent d'identifier et de localiser les neuf points considérés (comme SS6DPP). Lors de l'inférence, ces cartes permettent d'identifier et localiser les neuf points (ou moins si occultés) pour chaque instance d'objet. Les 8 cartes de directions pointent en chaque pixel d'un des huit coins vers le centre de l'instance d'objet associé à ce pixel. Lors de l'inférence, elles permettent d'associer chaque valeur de la carte de chaleur à un seul centre et donc à une seule instance d'objet.

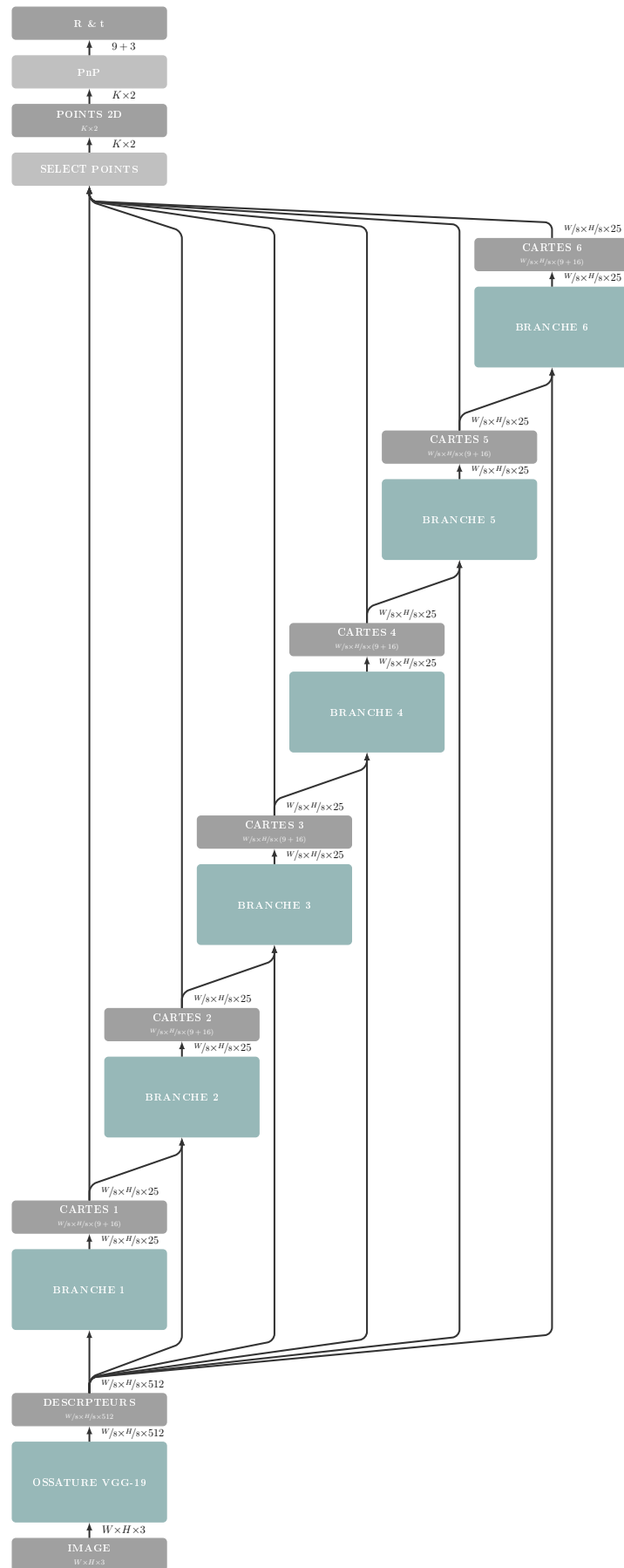


FIGURE 4.15 – Architecture complète de DOPE. Les branches 2 à 6 sont optionnelles.

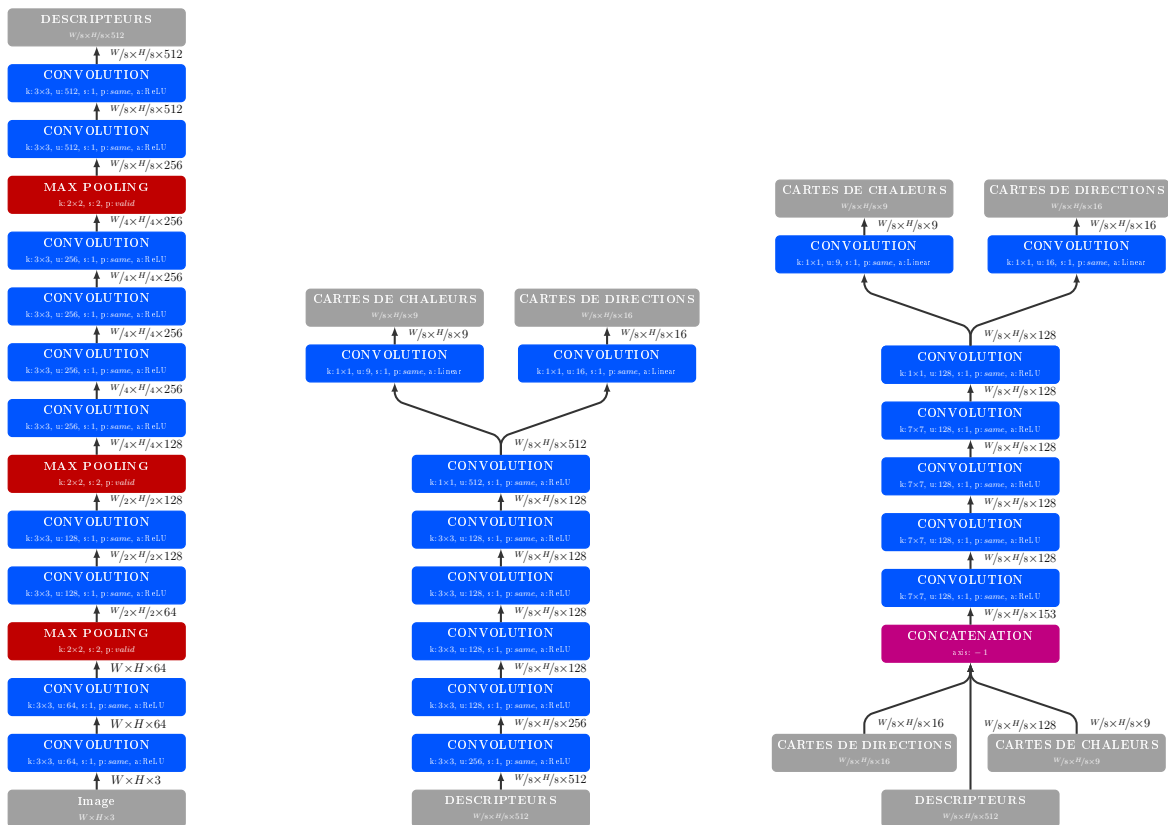


FIGURE 4.16 – Architecture des modules de DOPE

Cette méthode utilise une version complètement convolutionnelle de VGG de façon à pouvoir traiter des images de taille arbitraire et obtenir des prédictions dans le plan image. Les dix premières couches de convolution de VGG-19 sont utilisées pour extraire des descripteurs. Avec ces couches, trois couches de réductions spatiales ("max pooling") sont utilisées. La résolution des cartes de chaleur obtenues est donc de $W/s \times H/s$ pour une image de taille $W \times H$. Par la suite, des couches de convolution sont utilisées pour prédire les neuf cartes de chaleurs et les huit cartes directionnelles. L'architecture complète est détaillée à la figure 4.16. Après avoir inféré les cartes de chaleur et de directions, les coordonnées des points visibles de chaque instance d'objet sont calculées. Si quatre points ou plus sont visibles alors PnP est appliqué pour estimer la translation et rotation de l'instance d'objet.

DOPE ne peut estimer la pose que d'une seule catégorie d'objet à la fois. Une instance différente du réseau est entraînée indépendamment pour chaque catégorie d'objet, i.e. les poids du réseau sont spécialisés pour cet objet. Pour entraîner le réseau, des rendus photo-réalistes des objets de YCB-Vidéo sont utilisés en plus des images couleur. Le jeu d'entraînement développé par les mêmes chercheurs grâce à l'Unreal Engine est appelé FAT pour "Falling Things" [116]. DPOD obtient 82.98 % de précision ADD-3D sans affinage de la pose sur YCB-VIDÉO, et 95.15 avec. Sur LINEMOD OCCLUDED la méthode obtient 32.79 % de précision ADD-3D sans affinage et 47.25 % avec.

Vote par Pixel

La précision de DOPE est limitée par la résolution des cartes de chaleur et de direction qu'il produit. Pour obtenir des prédictions à la même résolution que l'image d'entrée, Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao et Xiaowei Zhou ont proposé d'utiliser une architecture inspirée de U-Net [97] nommée Réseau de Pixel Électeurs (Pixel-Voting Network, i.e. PV-Net) [86]. Cette dernière utilise des déconvolutions pour produire des prédictions en chaque pixel. Au lieu de prédire des cartes de chaleur pour chaque point, la méthode utilise ici huit cartes indiquant la direction de huit points sélectionnés à la surface du modèle 3D (et non sa boîte encadrante). Les points sont choisis pour être les plus loin les uns des autres pour assurer qu'au moins quatre points seront toujours visibles, peu importe le point de vue et les occlusions. Le réseau prédit aussi des masques de segmentation pour les catégories d'objets considérées, plus l'arrière-plan. Une fois les cartes de direction obtenues, un vote type RANSAC [36] permet de calculer efficacement les positions probables des huit points considérés pour chaque instance d'objet. Après quoi, PnP permet de retrouver la translation et la rotation. La figure 4.17 illustre la méthode.

PV-Net obtient 99 % de précision ADD-2D et 86.27 % d'ADD-3D-S sur LINEMOD. Sur LINEMOD OCCLUDED la méthode obtient 61.06 % d'ADD-2D et 40.77 % d'ADD-3D-S. Sur YCB-VIDÉO elle obtient 47.4 % d'ADD-2D et 73.4 % d'air sous la courbe ADD-3D-D.

Estimation Hybride de la Pose

Pour augmenter la robustesse et adresser le problème de symétrie d'objet Chen Song, Jiaru Song et Qixing Huang ont proposé en 2020 de prédire en plus des points d'intérêt, des arêtes entre les points ainsi que des correspondances entre pixels provenant de points symétriques sur le modèle 3D de l'objet. Vu que leurs méthodes infèrent plusieurs modalités, ils l'ont appelée Pose Hybride [109]. Les arêtes et les symétries contribuent à stabiliser l'entraînement quand les points d'intérêt sont ambiguës.

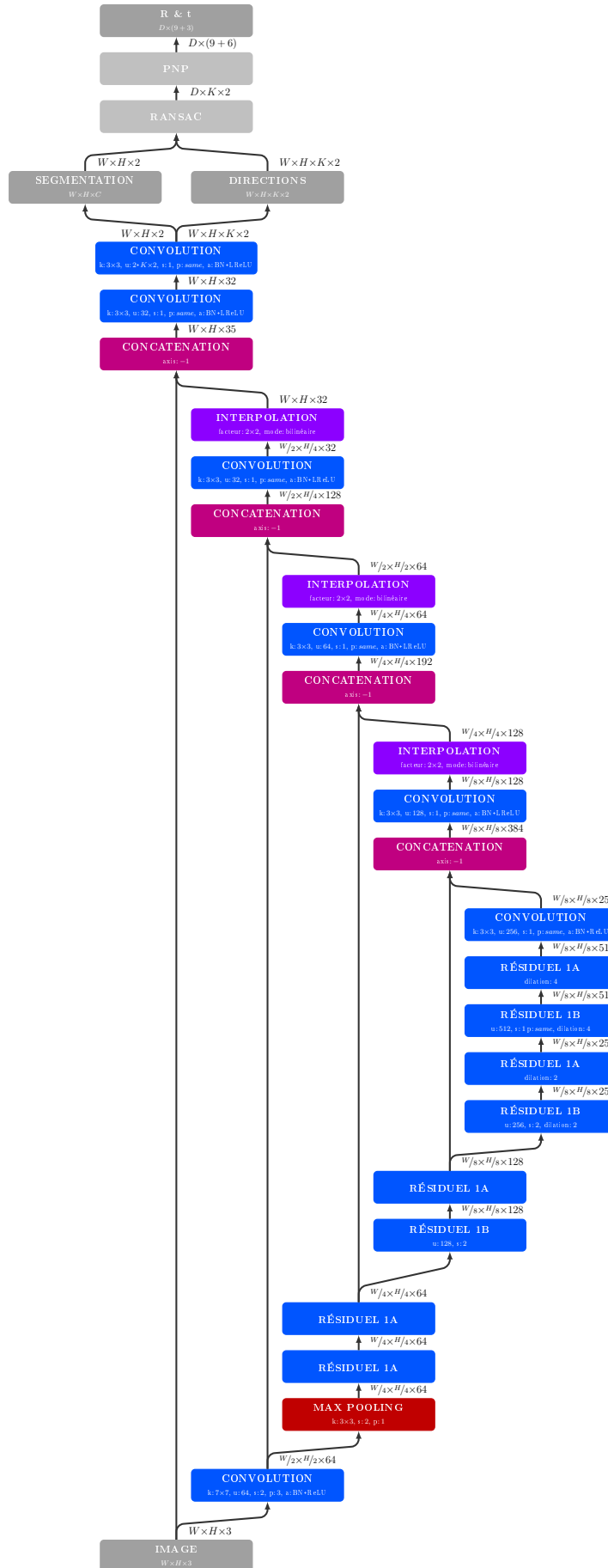


FIGURE 4.17 – Architecture de PV-Net [86].

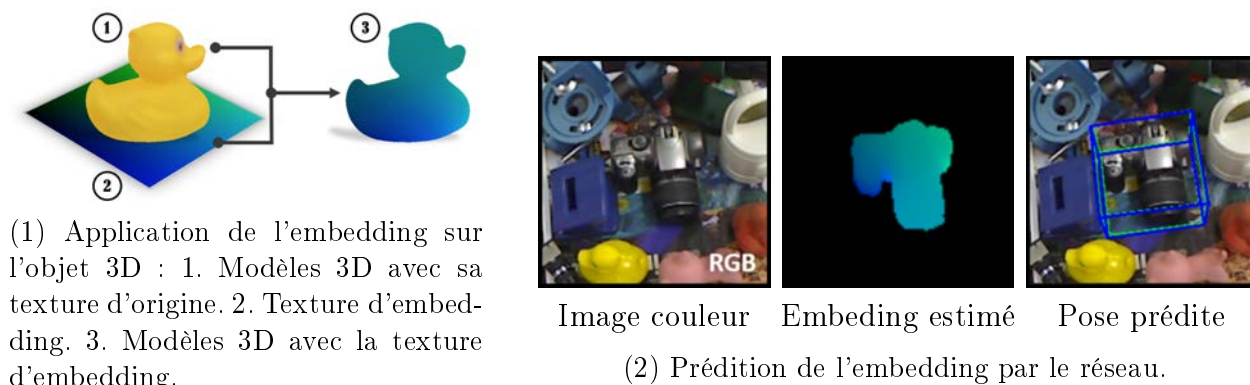


FIGURE 4.18 – Illustration de l'utilisation de l'embedding comme étiquette de la pose dans la méthode DPOD [129]. Les objets sont retexturés de façon à ce que chaque sommet se voit attribuer une couleur unique. Des rendus sont calculés à la pose étiquette pour créer les étiquettes des cartes UV. Les cartes obtenues sont utilisées comme étiquettes, et avec les images associées, elles sont utilisées pour entraîner le réseau à prédire les cartes U, V et le masque de segmentation.

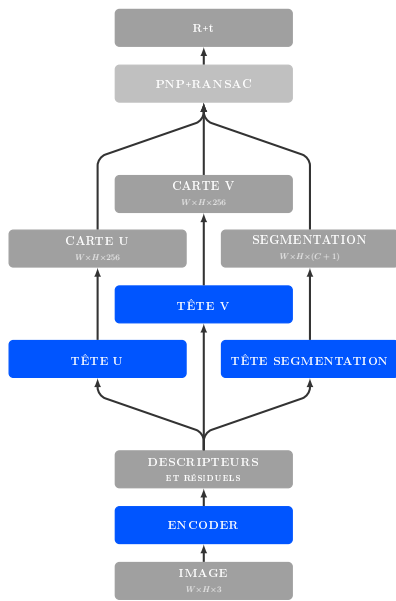
4.3.6 Embedding 3D

PV-Net prédit des cartes complexes avec un réseau de neurones profond. Ransac est intégré à l'architecture pour optimiser les calculs et écarter les prédictions aberrantes ("outliers"). Mais, à terme, seule huit points sont localisés.

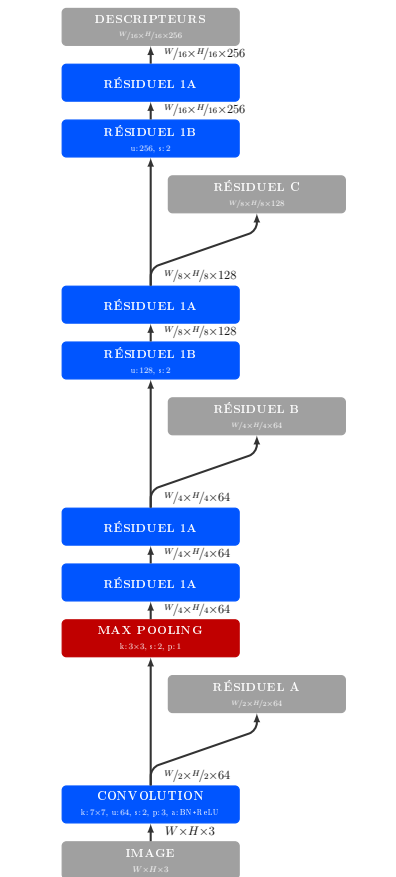
L'embedding est une technique qui permet de prédire des correspondances 2D-3D en chaque pixel. De ce fait, beaucoup plus de correspondances 2D-3D sont établies et l'étape statistique dispose donc de plus d'informations.

Détecteur Dense de Pose d'Objets

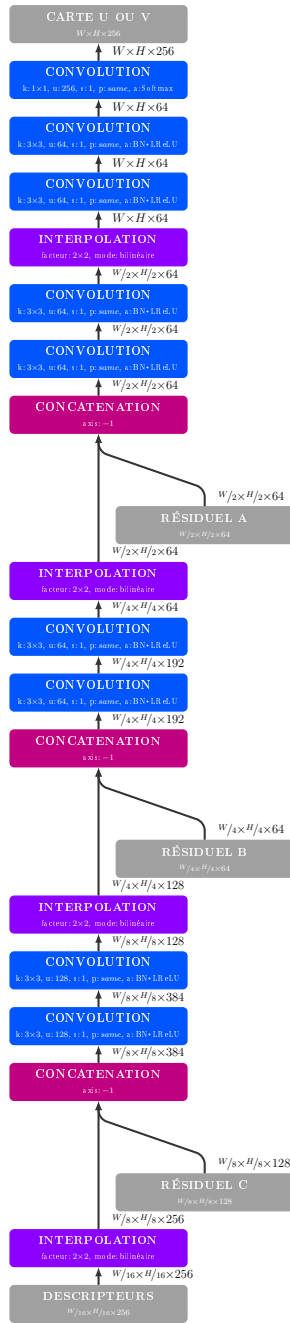
En 2019 Sergey Zakharov, Ivan Shugurov, Slobodan Ilic ont proposé le détecteur dense de pose d'objets ("Dense Pose Object Detector") [129]. L'idée est que chaque pixel appartenant à un objet soit associé à un point du modèle 3D. Mais comme cela a précédemment été montré, prédire directement des coordonnées en 3D ne fonctionne pas bien du fait de l'injectivité de la projection perspective. Pour contourner ce problème, chaque point à la surface du modèle 3D se voit attribuer une couleur unique. Le modèle 3D est re-texturé avec un "embedding", i.e. une "couverture" colorimétrique. Dans le cas de DPOD, deux canaux U et V chacun à 256 valeurs sont utilisés pour encoder 255^2 couleurs uniques. Le réseau, inspiré de ResNet [48], est utilisé pour prédire les canaux U et V et le masque de segmentation. L'architecture du réseau est présentée à la figure 4.19 et un exemple d'embedding à la figure 4.18. Les canaux sont prédits comme des probabilités d'appartenance à chaque couleur, le tenseur a donc une taille de $W \times H \times 256$. Prédire la couleur comme une valeur flottante comprise entre 0 et 255, 0 et 1 ou encore -1 et 1 ne fonctionne pas aussi bien. Le masque de segmentation a une dimension de $W \times H \times (C + 1)$. Une fois les cartes de couleurs UV et le masque de segmentation prédits, on dispose d'autant de correspondances 2D-3D que de pixels visibles de l'objet. PnP ne peut pas utiliser autant de points. Un échantillonnage basé sur RANSAC est utilisé pour filtrer les prédictions. Après quoi PnP peut être utilisé normalement pour obtenir la rotation et la translation de chaque objet.



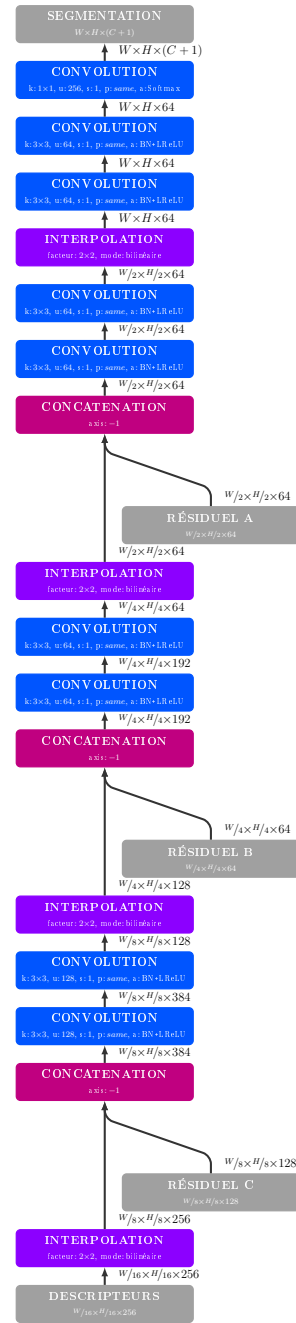
(1) Schéma global.



(2) Colonne d'extraction des descripteurs, aussi appelée encodeur.



(3) Tête de prédiction des canaux U ou V. Chaque canal est prédit indépendamment par une tête qui ne partage pas ses poids avec l'autre.



(4) Tête de prédiction du masque de segmentation. Elle ne partage pas ses poids avec les têtes de prédictions des canaux U ou V.

FIGURE 4.19 – Architecture de DPOD [129].

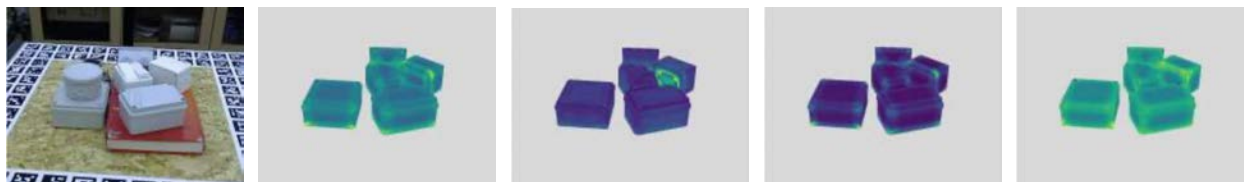


FIGURE 4.20 – Exemples d'embedding utilisés par CorNet [90].

Méthode	Ref.	ADD-2D	ADD-3D	ADD-3D-S
BB8	[91]		43.60 %	
DPOD	[129]		47.25 %	
YOLO 6D	[115]		55.95 %	
PV-Net	[86]	99.0 %		86.27 %
Pose-CNN+Deep IM	[126, 72]		88.60 %	
Hybrid Pose	[109]		91.30 %	
DPOD+Affinage	[129]		95.15 %	

TABLE 4.3 – Résultats d'estimation de la pose d'objets sur LINEMOD [50].

Détection d'Objets Inconnus

DPOD a montré que les embeddings sont capables de généraliser sur plusieurs objets. En 2020 Georgia Pitteri, Aurélie Bugeau, Slobodan Ilic et Vincent Lepetit ont montré que les embeddings peuvent généraliser sur des objets inconnus, c'est-à-dire qu'ils n'ont pas été utilisés pendant l'entraînement [90]. Pour ce faire, l'embedding calculé est modifié pour considérer la géométrie de l'objet localement dans un rayon de 3 cm, et non sur tout l'objet. De ce fait, si des objets partagent des caractéristiques géométriques, alors le réseau pourra prédire les embeddings, même s'il ne connaît pas ces objets. Pour prédire les embeddings, ils utilisent une version modifiée de U-Net qu'ils appellent CorNet. Ils entraînent le réseau grâce à des rendus photo-réalistes des objets de T-LESS calculés grâce à Blender. Le jeu de données synthétiques est appelé "Synthétic T-LESS".

Conclusion Sur la Détection 3D

Les progrès réalisés en classification d'images grâce aux réseaux de neurones convolutionnels n'ont pas pu être utilisés directement pour estimer la pose des objets. Les approches qui ont essayé d'utiliser directement les CNNs pour estimer les six degrés de liberté ont fait face au problème de la projection perspective et de l'ambiguïté avec l'échelle des objets. PoseNet produit par exemple des résultats avec des erreurs d'estimation de la distance à l'objet.

Traditionnellement, l'estimation de la pose était basée sur la reconnaissance et la localisation de patrons, comparables à des parties d'objets de DPM. Localiser ces parties sur l'image est aussi suffisant pour retrouver la pose de l'objet grâce à l'algorithme PnP qui s'occupe du problème de la projection inverse. C'est seulement quand les CNNs ont évolué en réseaux complètement convolutionnels qu'il a été possible de localiser avec précision des parties d'objet et de résoudre les problèmes de détection 2D et 3D avec des CNNs. D'ailleurs, les CNNs ont pour la première fois apporté une solution temps réel à ces problèmes.

Par la suite, les encodeurs-décodeurs ont permis d'augmenter la précision en fournissant plus de correspondances 2D-3D robustes aux occlusions. L'externalisation de la fusion d'informations grâce à RANSAC a permis de consolider cette robustesse aux occlusions.

Conclusion

Dans ce chapitre nous avons présenté les méthodes de détection les plus connues. Ces méthodes, toutes basées sur des réseaux de neurones, ont repoussé les limites du possible en matière de classification d'images et de localisation d'objets. Elles ont aussi rendu possible l'estimation de la pose d'objets à partir d'une image couleur seulement. Dans ce domaine, les données d'entraînement utilisées ont changé progressivement. Au lieu d'utiliser des images réelles, ce sont souvent des images synthétiques, des rendus 3D, qui sont utilisés.

Les performances des CNNs, comme toutes les méthodes d'apprentissage supervisé, sont dépendantes des données d'entraînement. Il est important que ces images soient nombreuses pour permettre à la descente de gradient de fonctionner et éviter la sur-spécialisation. Elles doivent être variées et être représentatives des conditions d'utilisation de la méthode en production. Par exemple, si des changements de luminosité ou des occlusions sont à prévoir, il est essentiel qu'ils soient présents dans les images. Les images synthétiques, dont la qualité a progressé, permettent de répondre à ces besoins.

Dans le prochain chapitre nous nous intéresserons aux jeux de données utilisés pour entraîner et valider les résultats de détections 3D des méthodes de l'état de l'art.

Chapitre 5

Jeux de données

Introduction

Pour étudier les méthodes de détection 3D, il est important de bien analyser les jeux de données. Les méthodes de l'art utilisent des réseaux de neurones. Elles dépendent donc entièrement des données d'apprentissage. La capacité du réseau à généraliser avec succès sur de nouvelles images nécessite que les images et les poses vues pendant l'entraînement soient les plus nombreuses et variées possibles. Différents jeux de données offrent différents enjeux et proposent d'évaluer les performances des méthodes face à certains facteurs de difficulté. La variété des images assurera aussi que la méthode est robuste et testée contre ces facteurs. Il est aussi important d'utiliser les mêmes jeux de données d'une méthode à l'autre avec le même protocole de préparation des images pour pouvoir comparer ces méthodes.

Nous cherchons à mettre en place une application d'aide à la maintenance, ou d'aide à l'assemblage d'objets et de machines à utilisation industrielle. Le scénario caractéristique comporterait donc un ou plusieurs objets se trouvant en face de la caméra à une distance permettant de le saisir. Plus particulièrement, on s'intéresse à de petits objets de l'ordre du centimètre, disposés sur table avec le manipulateur (i.e. la caméra) se trouvant devant la table.

Nous avons trouvé plusieurs jeux de données qui correspondent à notre cas, et qui sont utilisés par les méthodes de l'état de l'art. La première section est consacrée au jeu de données LINEMOD qui est la référence du domaine. Puis, nous présenterons l'extension OCLUDED de LINEMOD qui propose des occlusions extrêmes. Ensuite, nous détaillerons T-LESS qui figure des objets à caractère très industriel. Nous en viendrons ensuite au jeu de données YCV-VIDÉO qui a été récemment proposé spécialement pour entraîner des réseaux de neurones. Les sections suivantes sont consacrées à d'autres jeux de données qui sont rarement utilisés dans l'état de l'art : ITODD, HOMEBREW, FALLING THINGS, RU-APC, IC-BIN, IC-MI, T-LESS Synthétique et enfin le challenge BOP. Ce chapitre présente ces jeux de données et détaille leurs protocoles d'acquisition, la qualité de leurs étiquettes, leurs facteurs de difficulté et leurs limitations.

5.1 LINEMOD

LINEMOD est un jeu de données proposé par Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige et Nassir Navab en 2012 [50]. Il a été constitué dans le cadre de leurs travaux de détection 3D. Leur méthode LINEMOD utilise des patrons combinant des gradients calculés sur les images couleur et vecteurs normaux calculés sur les cartes de profondeur. Ce jeu de données est donc antérieur à l'air de



FIGURE 5.1 – Image couleur et carte de profondeur du jeu de données LINEMOD. On remarque la présence du plateau avec les marqueurs ArUco et les autres objets du jeu de données qui ne sont pas annotés dans cette image.




Objet			
Ape	1236	non	non
Benchviseblue	1215	oui	oui
Bowl	1233	oui	non
Cam	1201	non	oui
Can	1196	oui	non
Cat	1179	non	non
Cup	1240	oui	non
Driller	1188	oui	oui
Duck	1254	oui	non
Eggbox	1253	oui	non
Glue	1220	non	oui
Holepuncher	1236	oui	non
Iron	1152	oui	oui
Lamp	1227	oui	non
Phone	1243	non	oui
Total	18 273		

TABLE 5.1 – Information générale sur le jeu de données LINEMOD. De gauche à droite : nombre d'images, symétries, modèle 3D texturé.

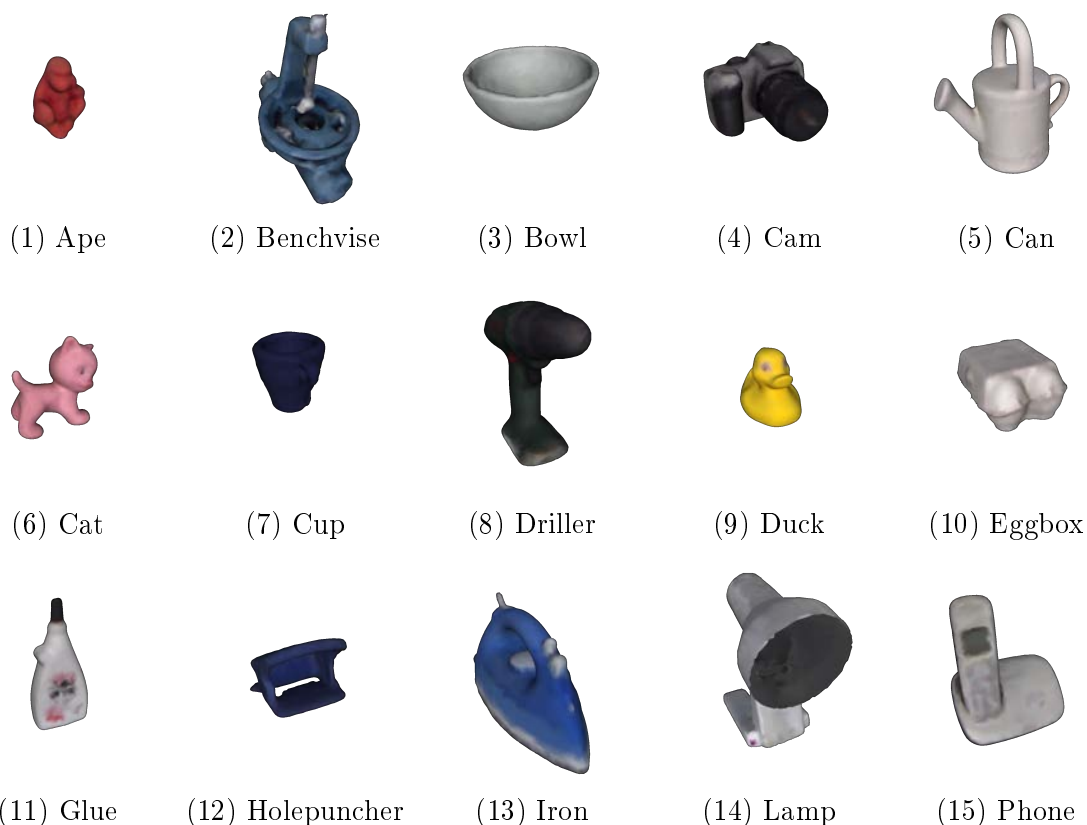


FIGURE 5.2 – Objets du jeu de données LINEMOD. Les 15 modèles 3D sont reconstruits à partir d'images qui ne sont pas distribués avec le jeu de données. Les pixels appartenant à un objet sont re-projeté dans l'espace 3D de l'objet. Cet espace est ensuite voxelisé. Les voxels sont ensuite convertis en un maillage triangulaire.

l'apprentissage par réseau de neurones profonds.

5.1.1 Informations générales

Le jeu de données est composé de quinze séquences, toutes de plus de 1100 images. Chaque séquence présente un des quinze objets dont la pose est étiquetée. Les modèles 3D des objets sont fournis sous la forme de nuages de points (format XYZ). Pour dix des objets, un maillage PLY couleur reconstruit est aussi inclus. Les images et les cartes de profondeur sont photographiées avec une Kinect dont les paramètres intrinsèques sont fournis. Dans chaque image, la position et l'orientation d'un objet sont annotées. Le tableau 5.1 regroupe ces quelques informations et précise pour chaque séquence combien d'images sont photographiées.

5.1.2 Protocole d'acquisition

Voici ce que nous disent les auteurs dans leur article en Anglais :

“For comparison, we created a large dataset of 15 registered video sequences of 15 texture-less 3D objects. Each object was stuck to the center of a planar board with markers attached to it, for model and image acquisition. The markers on the board provided the corresponding ground truth poses. Each object was reconstructed first using a set of images and the corresponding poses using a simple voxel based approach. After reconstruction, we added close range and far range 2D and 3D clutter to the scene and took the evaluation sequences. Each sequence contains more than 1,100 real images from different view points. In order to guarantee a well distributed pose space sampling of the dataset pictures, we uniformly divided the upper hemisphere of the objects into equally distant pieces and took at most one image per piece. As a result, our sequences provide uniformly distributed views from 0 – 360 degree around the object, 0 – 90 degree tilt rotation, 65 – 115 cm scaling and 45 degree in-plane rotation.”

Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes [50], Section 4, Page 8.

Et voici une explication en français. Ce jeu de données a été constitué en deux temps :

1. Reconstructions des modèles 3D des objets à partir de photographies.
2. Acquisition des images pour l'évaluation et les tests.

Reconstructions 3D des objets

Dans un premier temps, pour chacun des 15 objets, des photographies où figure un seul objet sont prises. L'objet est placé au centre d'un plateau comportant plusieurs marqueurs. La position relative de l'objet par rapport au plateau est connue. Les marqueurs permettent de calculer la position du plateau par rapport à la caméra, par exemple, grâce à une technique comme ArUco [39] qui est implémentée dans OpenCV [84]. Une fois la pose du plateau calculée, on peut donc en déduire la pose de l'objet par rapport à la caméra. Ce qui permet de calculer des rendus 3D de l'objet et d'obtenir son masque de segmentation. En calculant l'inverse de la projection perspective, on peut ensuite reprojeter les pixels appartenant à l'objet dans le repère objet. On obtient ainsi un nuage de points dont les couleurs sont connues.



FIGURE 5.3 – Image de LINEMOD où l’on aperçoit l’ordinateur qui gère l’acquisition des images. On y voit l’outil d’acquisition implémenté par les auteurs. Probablement en Matlab. Cet outil permet de visualiser la position de la caméra par rapport au plateau ainsi que les points de vue déjà acquis.

Des modèles surfaciques sont aussi fournis. Pour les obtenir, le nuage de points est converti en voxels (pixels en 3 dimensions). Ces voxels sont ensuite convertis en maillage de triangles. Les modèles 3D obtenus sont illustrés en figure 5.2.

Acquisition des images pour l’évaluation

Dans un second temps, d’autres objets sont ajoutés sur le plateau pour créer des occlusions et une nouvelle série de photographies est prise. Un exemple avec le canard est présenté en figure 5.1. Les prises de vues sont réparties le plus équitablement possible dans une demi-sphère imaginaire posée au dessus du plateau. Pour s’aider dans l’acquisition des images, les auteurs ont implémenté un outil Matlab qui affiche la position de la caméra par rapport au plateau et aux points de vue déjà photographiés. Cet outil est visible dans certaines images comme le montre la figure 5.3. De la même façon que précédemment, la pose de l’objet est calculée grâce aux marqueurs présents sur le plateau. Mais pas celles des autres objets puisqu’ils ne sont pas référenciés par rapport au plateau et ne sont pas considérés par les auteurs. Les auteurs ont photographiés de l’ordre de 1100 points de vue pour chacun des 15 objets.

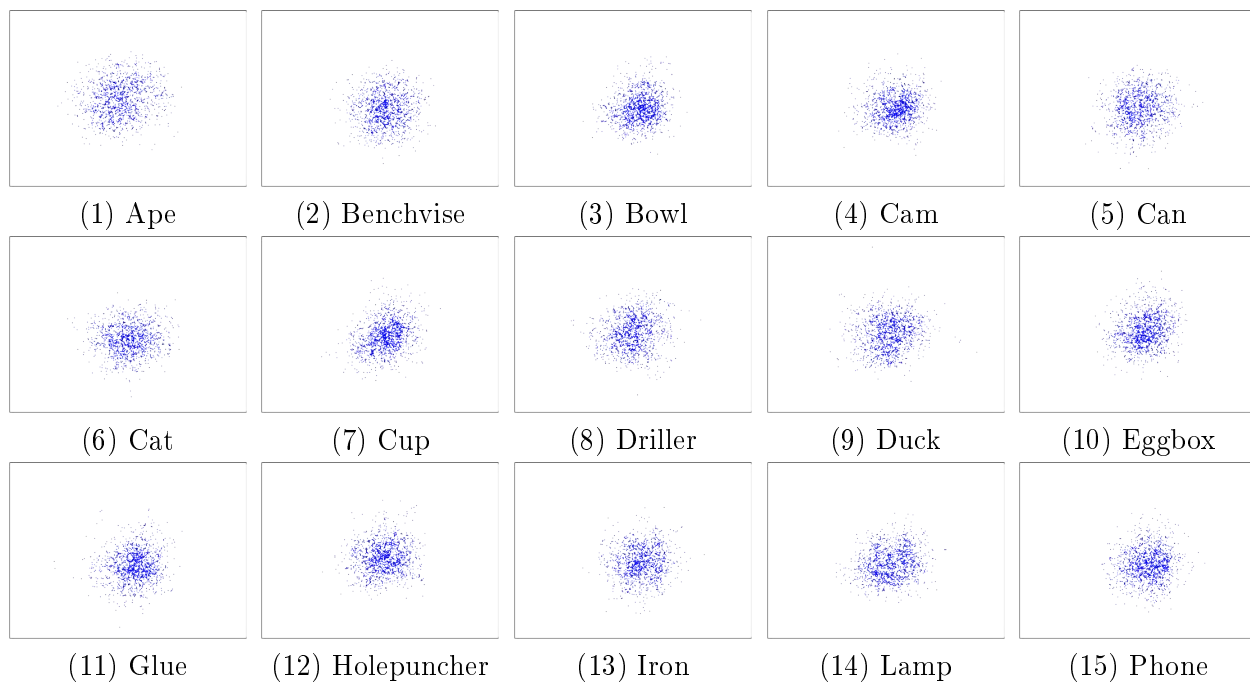


FIGURE 5.4 – Positions où apparaissent les objets dans les images de LINEMOD. Les points sont obtenus en projetant l’origine du repère objet dans le plan image. On voit un fort biais central pour tout les objets. La caméra est donc braquée en direction de l’objet au centre du plateau dans la plupart des cas.

5.1.3 Protocole d’utilisation

Ce jeu de données contient des modèles 3D et des images. Dans l’article des auteurs [51, 50], les modèles 3D sont utilisés pour l’entraînement et les images pour la validation et les tests.

Néanmoins, on voit de plus en plus souvent dans la littérature des personnes qui divisent les images en deux groupes : une partie pour l’entraînement, et une partie pour les tests et la validation. La séparation se fait soit de façon à échantillonner des poses de test de façon à ce qu’elles soient les plus réparties possible, soit de façon aléatoire. Dans les deux cas la proportion communément utilisée est de 20 % pour l’entraînement et 80 % pour les tests ce qui laisse très peu d’images d’entraînement.

5.1.4 Intérêt et points forts

LINEMOD propose d’abord une bonne variété d’objets. Certains sont texturés comme le mug, le bol, le fer à repasser ou la perceuse mais la plupart ne le sont pas comme le singe, le chat ou encore la lampe qui est entièrement blanche. L’absence de texture implique qu’il y a peu de caractéristiques de couleur ou de surface qui permettent de discriminer ces objets. En quelque sorte, ils sont moins accrocheurs visuellement. C’est un facteur de difficulté pour beaucoup de méthodes se basant sur des descripteurs visuels comme SIFT.

Les points de vue sont variés et présentent un scénario sur table qui correspond à notre cadre applicatif, aussi bien par la distance à l’objet qui est comprise entre 0.6 et 1.10 mètres, que par la répartition des points de vue autour des objets. Comme indiqué dans le protocole d’acquisition, les points de vue sont répartis sur une demi-sphère au dessus de l’objet. Les points de vue des objets sont présentés en figure 5.5. Ils sont extrêmement bien répartis pour une caméra sans doute posée sur trépied ou portée à la main.

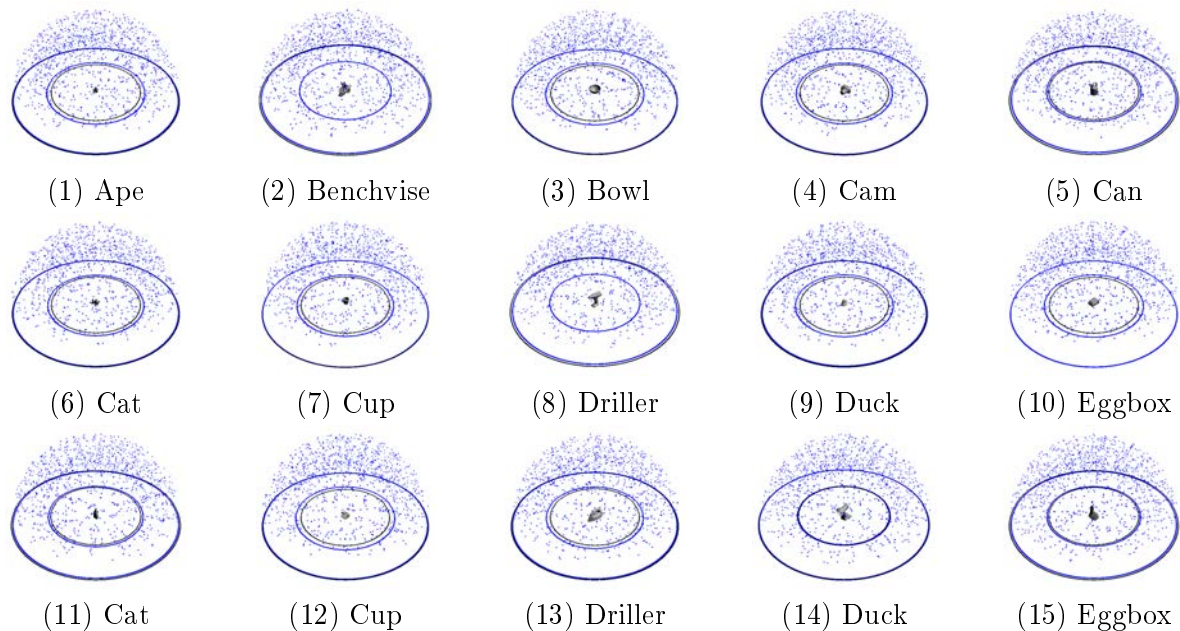


FIGURE 5.5 – Positions où se trouvait la caméra lorsqu'elle a photographié les objets de LINEMOD. Chaque point bleu correspond à la projection de l'origine du repère de la caméra dans le repère objet. La projection est obtenue en inversant la matrice de projection $P_{i,j,M \rightarrow I}$ pour toute les images.

Après étude du jeu de données, on remarque que la caméra est souvent braquée sur l'objet au centre du plateau comme le montre la figure 5.4. Il y a donc un fort biais central dans les images, ce qui est intéressant dans notre cas. En effet, les casques de réalité mixte comme le Microsoft HoloLens 2® [77] utilisent des traqueurs oculaires pour l'interaction. Et il se trouve que la vision humaine a, elle aussi, un fort biais visuel.

Autre point intéressant, le jeu de données est photographié à l'aide d'une caméra grand public dont le SDK est disponible en ligne et multi-plateformes. La caméra produit des images en couleur mais aussi des cartes de profondeur. Ces cartes sont particulièrement intéressantes pour la détection 3D, car elles permettent de lever l'ambiguïté lors de la reconstruction 3D de la scène, et aident beaucoup les méthodes de détection 3D.

Et enfin, les annotations sont obtenues automatiquement grâce aux marqueurs présents sur le plateau. Les images ne sont pas annotées manuellement. La méthode étant très stable, les étiquettes sont de très bonne qualité.

Tous ces points font de LINEMOD un jeu de données particulièrement intéressant.

5.1.5 Limitations

La méthode de détection 3D utilisée par les auteurs est entraînée sur des données synthétiques : des rendus couleurs et des rendus de cartes de profondeur obtenus grâce aux modèles 3D. Les images réelles sont destinées uniquement à servir pour la validation et les tests de performance. Le jeu de données ne contient aucune image réelle d'entraînement.

Pour entraîner des réseaux de neurones sur des images réelles, il faut diviser les images en deux groupes comme mentionné précédemment. Néanmoins, les images sont si peu nombreuses qu'il faut augmenter synthétiquement leur nombre pour satisfaire aux besoins d'un réseau de neurones. Par exemple avec des variations de contrastes, de luminosités, de teintes, et de saturations. Les méthodes qui prédisent des points 2D au lieu de régresser directement la pose 3D peuvent aussi redimensionner, recadrer et transformer les images comme elles le souhaitent puisque ces modifications peuvent être prise en compte dans le calcul de leurs

Objets	$min(v)$	$max(v)$	$\mu(v) \pm \sigma(v)$
Ape	68.45 %	100.00 %	98.26 \pm 2.65 %
Benchvisеblue	71.00 %	099.63 %	94.90 \pm 2.80 %
Bowl	73.44 %	100.00 %	94.86 \pm 4.49 %
Cam	77.07 %	100.00 %	97.64 \pm 2.82 %
Can	77.68 %	099.79 %	96.51 \pm 2.80 %
Cat	73.14 %	099.93 %	97.43 \pm 2.60 %
Cup	80.40 %	100.00 %	96.36 \pm 3.01 %
Driller	72.98 %	100.00 %	96.96 \pm 3.63 %
Duck	73.26 %	100.00 %	97.49 \pm 2.77 %
Eggbox	65.85 %	100.00 %	96.90 \pm 3.47 %
Glue	69.35 %	100.00 %	97.94 \pm 3.40 %
Holepuncher	71.31 %	100.00 %	96.19 \pm 3.43 %
Iron	70.83 %	099.76 %	93.71 \pm 4.77 %
Lamp	56.72 %	099.70 %	94.59 \pm 4.70 %
Phone	77.43 %	100.00 %	97.23 \pm 3.14 %
Touts	56.72 %	100.00 %	96.47 \pm 3.68 %

TABLE 5.2 – Visibilités minimums, maximums, moyennes et écarts types des objets du jeu de données LINEMOD. La visibilité est calculée comme le rapport d’air entre masque visible et masque complet. Le masque complet est celui de tout l’objet, même occulté.

Objets	$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
Ape	1.96 \pm 6.67 mm	4.44 \pm 5.36 mm	3.82 %
Benchvisеblue	1.49 \pm 8.63 mm	5.40 \pm 6.89 mm	7.51 %
Bowl	2.47 \pm 7.60 mm	5.15 \pm 6.11 mm	10.12 %
Cam	2.21 \pm 7.61 mm	5.05 \pm 6.11 mm	14.76 %
Can	2.82 \pm 8.21 mm	5.52 \pm 6.70 mm	9.56 %
Cat	0.03 \pm 6.87 mm	3.96 \pm 5.61 mm	4.10 %
Cup	0.88 \pm 7.95 mm	5.06 \pm 6.20 mm	7.33 %
Driller	2.81 \pm 5.69 mm	4.71 \pm 4.25 mm	7.77 %
Duck	1.68 \pm 8.31 mm	4.98 \pm 6.86 mm	6.47 %
Eggbox	1.39 \pm 6.74 mm	4.65 \pm 5.07 mm	1.71 %
Glue	3.13 \pm 6.63 mm	5.31 \pm 5.06 mm	5.41 %
Holepuncher	1.21 \pm 9.49 mm	5.62 \pm 7.75 mm	6.21 %
Iron	-0.33 \pm 8.52 mm	5.38 \pm 6.61 mm	11.27 %
Lamp	-0.64 \pm 6.86 mm	4.51 \pm 5.20 mm	5.39 %
Phone	3.20 \pm 8.03 mm	5.88 \pm 6.34 mm	8.53 %
Touts	1.49 \pm 7.75 mm	5.07 \pm 6.05 mm	7.30 %

TABLE 5.3 – Erreurs mesurées dans les cartes de profondeurs du jeu de données LINEMOD. L’erreur est calculée pixel-à-pixel entre la carte de profondeurs photographiée et la carte de profondeur synthétisée à la pose étiquetée de l’objet. Les pixels hors du masque de l’objet sont ignorés. Les pixels pour lesquels les différences sont supérieures à 50 mm sont ignorés et considérés comme des valeurs abérantes. La première colonne présente les erreurs moyennes et les écarts types associés. La deuxième, les erreurs moyennes et les écarts types associés mais avec des erreurs calculées en valeurs absolues. La dernière rapporte le pourcentage de pixels ignorée selon la règle des 50 mm.

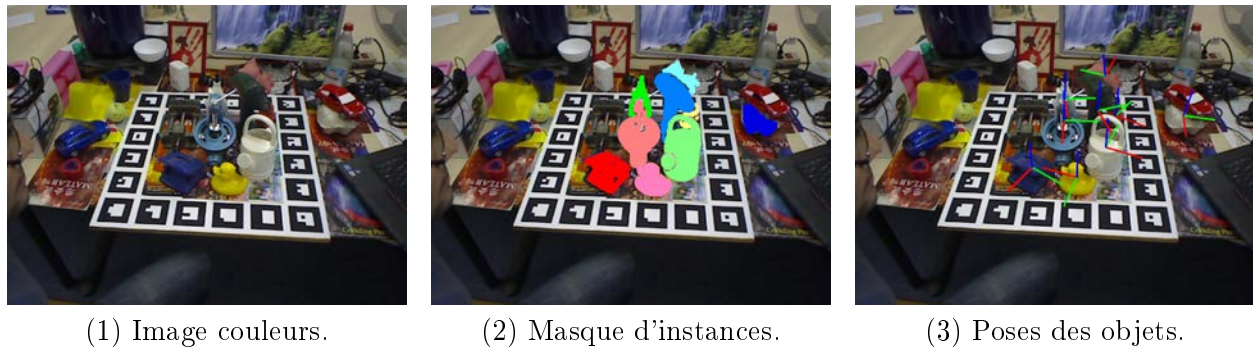


FIGURE 5.6 – Image du jeu de données LINEMOD+OCCLUDED

étiquettes. Ce qui leurs apporte d'ailleurs peut-être un avantage.

Autre soucis, un seul objet est annoté dans chaque image. Tous les autres objets qui sont présents dans les images ne sont donc pas utilisables. Ce problème s'accroît quand un objet non-annoté vient occulter l'objet au centre du plateau. Lorsque cela se produit, rien ne nous permet de savoir directement dans les étiquettes que l'objet est occulté. Les pixels appartenant à l'objet occultant vont donc être considérés comme appartenant à l'objet occulté. Il est possible grâce aux cartes de profondeur de retrouver un masque de la partie non-occultée de l'objet. Mais cette technique utilise un seuil fixé manuellement pour identifier les changements de profondeurs. Si on a une transition "douce" entre deux objets dans la carte de profondeur, alors la frontière entre les deux objets sera imprécise.

Le fait qu'un seul objet soit annoté pénalise les méthodes qui estiment la pose de plusieurs objets d'une passe, comme YOLO6D [115]. La majeure partie de leurs étiquettes sont vides (valeurs non-utilisées) ce qui ralentit ou déstabilise leurs entraînements.

5.2 LINEMOD+OCCLUDED

LINEMOD+OCCLUDED [9] est une extension LINEMOD [50]. Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, Carsten Rother ont manuellement annoté la pose de plusieurs objets visibles de la deuxième séquence de LINEMOD consacrée à la vise d'établie ("*Benchvise*"). Ne cherchez pas des informations sur ce jeu de données dans le corps de leur publication intitulée "*Learning 6D Object Pose Estimation Using 3D Object Coordinates*", il n'y en a aucune. Ce travail d'annotation sur LINEMOD est présenté dans les ressources supplémentaires disponibles sur le site de Springer. Les données sont disponibles sur le site de l'Université de Heidelberg. Mais les auteurs recommandent maintenant de télécharger les données depuis la page du Challenge BOP, car certaines annotations ont été corrigées (Téléchargez "*All Test Images*").

5.2.1 Informations générales

Ce jeu de données reprend la séquence *Benchvise* de LINEMOD et y ajoute les annotations pour les sept objets *Ape*, *Cam*, *Can*, *Cup*, *Driller*, *Duck*, *Eggbox* et *Glue*. La figure 5.6 présente une image issue de cette séquence avec les annotations supplémentaires. Pour chacun de ces objets, on a donc 1215 annotations supplémentaires. Grâce à ces annotations, les quelques ambiguïtés d'occlusions existantes dans LINEMOD, pour cette séquence, sont levées. De nombreuses occlusions annotées sont aussi ajoutées, certaines très prononcées.

5.2.2 Protocole d’acquisition

Les images et les modèles 3D sont ceux du jeu de données LINEMOD (voir Sous-Section 5.1.2). Pour ce qui est des annotations supplémentaire propre à LINEMOD+OCCLUDED voici ce que l’on peut lire dans les suppléments de l’article [9] :

“Our dataset and the dataset of [50] are free of occlusions. While the objects annotated in the dataset of [50] are embedded in dense clutter, they are still fully visible in each frame. Hence, to demonstrate robustness against occlusion we created a new dataset. We annotated one sequence (“Bench Vise”, ca. 1200 frames) of the dataset of [50] with 6DOF poses of 8 additional objects present in the scene. Depending on the viewing direction, these objects occlude each other to a large extent making this dataset very challenging. Fig. 7 shows one frame with all annotations marked, and a closeup of a heavily occluded object. We annotated the sequence by initializing the pose of each object by hand, and propagating the object pose via the groundtruth transformation of each frame. If the propagation produced errors or when the object was moved within the scene we reinitialized by hand. For each frame, all poses were refined by ICP. We term this dataset occlusion dataset and make the annotation data publicly available.”

Learning 6D Object Pose Estimation using 3D Object Coordinates -
Supplementary Material [9], Section 7.2, Page 13.

L’objectif est annoncé dès le début : obtenir un jeu de données avec des occlusions. Pour ce faire les auteurs ont choisi, plutôt que de créer un jeu de données contenant des occlusions, de rajouter des annotations à un jeu de données existant : LINEMOD. Le procédé d’annotation est semi-automatique. En effet les images d’une séquence de LINEMOD sont photographiées proches les unes des autres en terme de distance de points de vue. De plus, elles montrent les mêmes objets, généralement aux mêmes emplacements sur la table et le plateau. Ils ont donc pu initialiser manuellement la pose de chaque objet dans une image, et propager les annotations automatiquement aux images suivantes. Si jamais un objet est déplacé d’une image à l’autre, la pose de cet objet est ré-annoté manuellement, et le processus reprend. Bien que partiellement automatique, ce travail d’annotation demande une supervision humaine pour chaque image. C’est donc un processus relativement fastidieux.

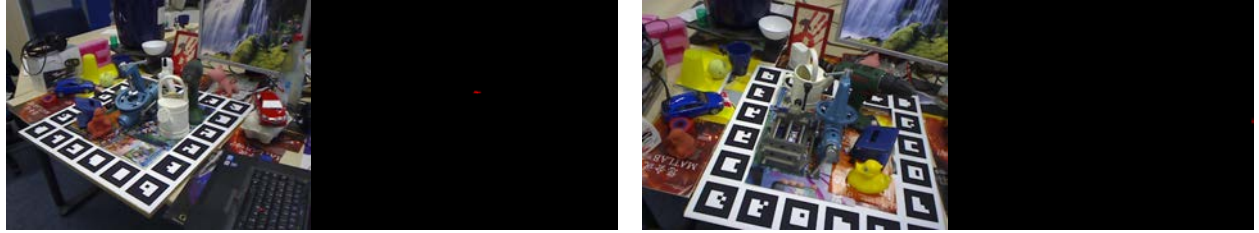
5.2.3 Protocole d’utilisation

Ce jeu de données, comme LINEMOD, est à utiliser pour des tests de performances ou de validations (c.f. Sous-Section 5.1.3). On rappelle que les auteurs de LINEMOD utilisent les modèles 3D et des rendus pour entraîner leur méthode. La méthode des auteurs de LINEMOD+OCCLUDED utilise aussi des rendus des modèles pour entraîner une forêt d’arbres décisionnels.

Dans la littérature récente LINEMOD+OCCLUDED n’est pas non plus utilisé pour l’entraînement. Les réseaux de neurones sont engrainés sur LINEMOD et grâce à des augmentations d’images qui synthétisent des occlusions, le CNN généralise sur LINEMOD+OCCLUDED. Des rendus photo-réalistes sont aussi utilisées pour l’entraînement.

5.2.4 Intérêts et points forts

OCCLUDED tient ses promesses en terme d’occlusions avec des objets presque complètement occultés. C’est particulièrement le cas du singe et du canard qui sont de petits objets



(1) Au centre de l'image, derrière l'arrosoir, le canard est présent et sa pose est annotée. Il s'agit ici d'une occlusion très importante. Certains objets de LINEMOD+OCCLUDED sont quasiment complètement occultés. Par exemple le Canard a la plus petite visibilité enregistrée : 0.05 %.

(2) Sur le bord droit de l'image, on aperçoit à peine la boîte d'œufs. Il s'agit ici d'un objet à la limite du hors champs. Certains objets de LINEMOD+OCCLUDED sont complètement hors champs.

FIGURE 5.7 – Exemples d'objets fortement occultés ou hors champs dans les images de LINEMOD+OCCLUDED.

Objets	$min(v)$	$max(v)$	$\mu(v) \pm \sigma(v)$	Hors champ
Ape	2.49 %	100.00 %	83.93 ± 23.31 %	0.00 %
Benchvisseblue	70.99 %	99.62 %	94.90 ± 02.80 %	0.00 %
Can	22.46 %	99.96 %	79.55 ± 16.16 %	0.00 %
Cat	1.44 %	100.00 %	64.04 ± 31.13 %	0.00 %
Driller	23.91 %	100.00 %	77.02 ± 16.76 %	0.00 %
Duck	0.05 %	100.00 %	80.76 ± 27.95 %	0.00 %
Eggbox	0.31 %	99.97 %	69.06 ± 27.16 %	0.34 %
Glue	0.14 %	100.00 %	70.57 ± 29.32 %	0.22 %
Holepuncher	25.43 %	100.00 %	91.27 ± 11.69 %	0.00 %
Tous	00.05 %	100.00 %	79.31 ± 24.23 %	0.06 %

TABLE 5.4 – Visibilités minimums, maximums, moyennes et écarts types des objets du jeu de données LINEMOD+OCCLUDED.

Objets	$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
Benchvisseblue	1.49 ± 8.63 mm	5.40 ± 6.89 mm	7.51 %
Ape	3.08 ± 8.42 mm	$5.94.72$ mm	10.10 %
Benchvisseblue	2.63 ± 6.23 mm	4.51 ± 5.04 mm	6.50 %
Can	4.48 ± 8.16 mm	6.31 ± 6.85 mm	12.49 %
Cat	4.02 ± 7.50 mm	6.21 ± 5.82 mm	21.15 %
Driller	3.45 ± 6.72 mm	5.76 ± 4.94 mm	11.43 %
Duck	4.58 ± 7.65 mm	5.82 ± 6.76 mm	11.75 %
Eggbox	3.86 ± 8.80 mm	7.35 ± 6.19 mm	6.68 %
Glue	2.26 ± 7.11 mm	5.30 ± 5.25 mm	22.48 %
Holepuncher	1.98 ± 8.36 mm	4.77 ± 7.14 mm	9.32 %
Tous	3.27 ± 7.52 mm	5.54 ± 6.05 mm	12.15 %

TABLE 5.5 – Erreurs mesurées dans les cartes de profondeur du jeu de données LINEMOD+OCCLUDED. La première ligne rappelle les erreurs de profondeur de la vis établie de LINEMOD sans les étiquettes de LINEMOD+OCCLUDED.

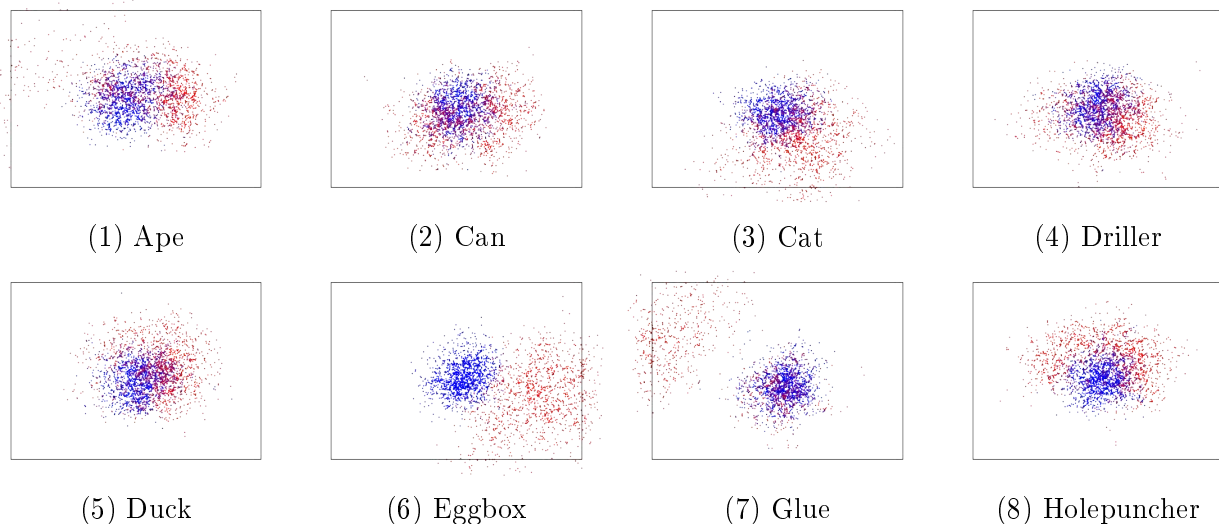


FIGURE 5.8 – Positions où apparaissent les objets dans les images de LINEMOD+OCCLUDED (en rouge) par rapport aux positions où ils apparaissent dans celles de LINEMOD (en bleu). Le rectangle gris délimite les bords de l'image. Tous les points en dehors correspondent à des objets partiellement ou totalement hors-champs.

et se retrouvent donc plus facilement masqués par les autres. Le Tableau 5.4 récapitule les visibilité minimums, maximums et moyennes des objets. L'objet le plus occulté est le tube de colle qui dans une image ne présente que 0.14 % de surface visible soit 99.86 % d'occlusions. Ces occlusions font du jeu de données OCCLUDED un jeu de données très difficile et parfait pour évaluer une méthode de détection face à ce facteur de difficulté.

Les étiquettes obtenus par les auteurs sont aussi de très bonnes qualités avec une erreur moyenne dans les cartes de profondeur de 3.27 *mm*. Le tableau 5.5 détaille l'erreur pour chaque objet. L'erreur moyenne en valeur absolue est de 5.54 *mm* et le pourcentage de pixels ignorés est de 12.15 %. La proportion de pixels ignorés a globalement augmenté. Une explication possible est que dans quelques images les étiquettes de certains objets soient imprécises. Mais si on compare seulement les pixels de la vis établie, alors on remarque que la proportion de pixels ignorés diminue. Les annotations apportés par LINEMOD+OCCLUDED ont donc permis de lever l'ambiguïté sur des occlusions précédemment non-identifiées. Il est possible que d'autres objets non annotés, même dans LINEMOD+OCCLUDED, contribuent à accroître la proportion de pixels ignorés, alors même que les étiquettes sont correctes comme c'était le cas pour la vis établie.

5.2.5 Limitations

Les objets annotés en plus dans OCCLUDED se trouvent autour de la vis établie. Par conséquent la caméra ne les braque pas. De ce fait ils apparaissent excentrés dans l'image, voire même en périphérie comme le montre la figure 5.8. Certaines fois, ils sont mêmes complètement hors-champs. Les poses étant propagées à partir des images précédentes grâce à une technique de suivi actif, la pose d'un objet hors-champ peut être connue.

Le fait que ces annotations soient présentes ne constituent en rien une erreur d'annotation. L'objet est bien présent. Si on cherche à évaluer une technique de suivi actif, ces étiquettes peuvent être intéressantes. Néanmoins il n'est juste pas possible de dire si l'objet est présent à partir d'une seule image ou, s'il est hors champ. Pour évaluer une technique de détection sur une image donnée, ces étiquettes peuvent (doivent) être ignorées.

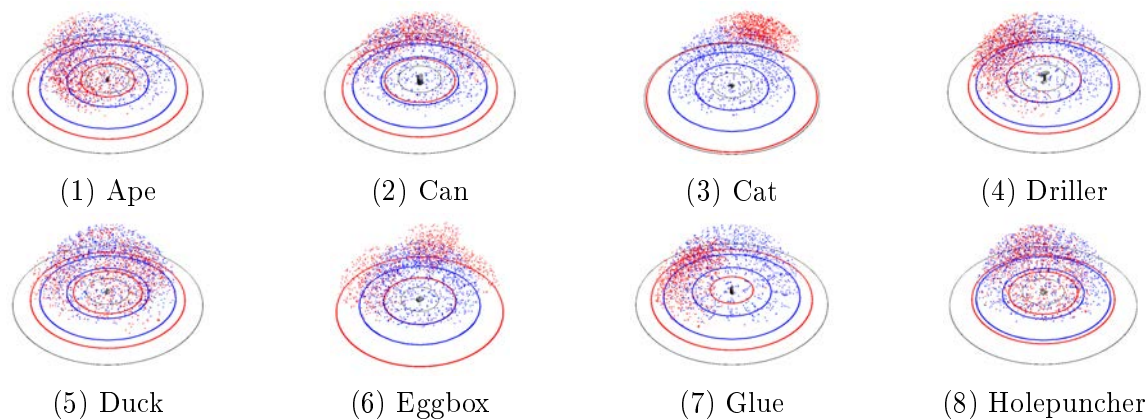


FIGURE 5.9 – Positions où se trouvait la caméra lorsqu'elle a photographié les objets de LINE-MOD+OCCLUDED (en rouge) par rapport aux positions où elle se trouvait quand elle a photographié les objets de LINEMOD (en bleu).

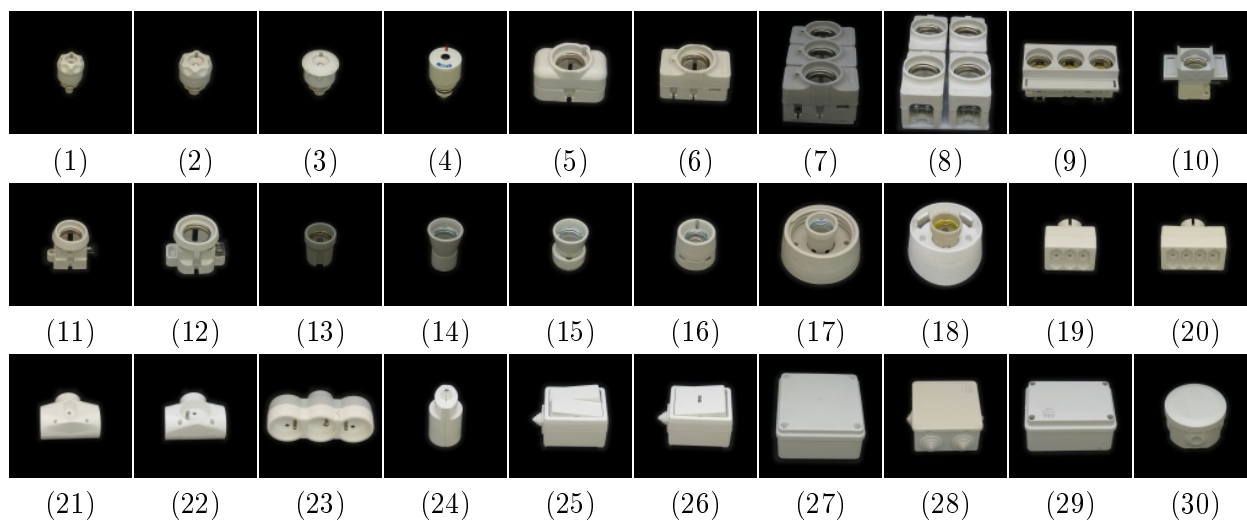


FIGURE 5.10 – Les 30 objets du jeu de données T-LESS [56]

De plus, la caméra est plus proche ou plus éloignée de ces objets comparé à LINEMOD, comme on peut le constater à la figure 5.9. Par exemple pour la boîte d'œufs 5.96. L'espace de pose est donc plus vaste et complexe comparé aux seules images de LINEMOD.

En somme LINEMOD+OCCLUDED est un jeu de données difficile à solutionner, surtout si on ne dispose que des images de LINEMOD comme données d'entraînement.

5.3 T-LESS

Le jeu de données "*texture-less*" [56], i.e. sans texture, a été réalisé entre 2015 et 2017 par Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, et Xenophon Zabulis.

5.3.1 Informations générales

Ce jeu de données se concentre sur des objets industriels comme des boîtes électriques ou des robinets de chauffage. Au total, trente objets sont photographiés avec trois caméras différentes : une Microsoft Kinect V2, une Primesense CARMINE 1.09 et une Canon IXUS

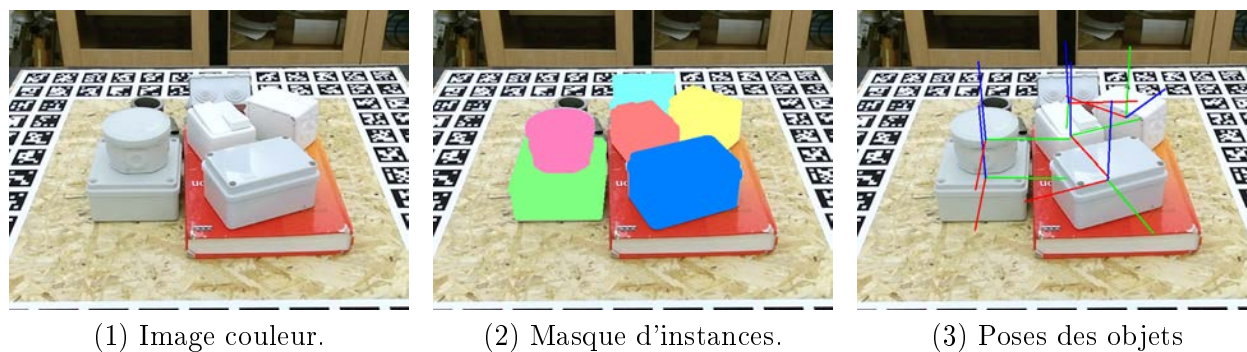


FIGURE 5.11 – Image du jeu de données T-LESS [56].

950 IS. Deux de ces caméras sont des caméras de profondeurs RGB-D, et la dernière une caméra RGB haute définition.

Les objets sont présentés dans la figure 5.10. Ils ont principalement une couleur unie bien que certains robinets aient des touches de couleurs. Si l'on compare les objets d'un même type, par exemple les robinets (objets 1 à 4), on remarque qu'ils présentent peu de détails permettant de les différencier entre eux. Ils sont aussi peu texturés au sens où il n'y a aucun motif, texte, logo, détail de surface, sur les objets. D'où le nom du jeu de données. Ils ont aussi beaucoup de symétries et certains objets sont même des sous-parties d'autres. Par exemple, l'objet 7 est composé de trois instances de l'objet 6.

Les modèles 3D dessinés sur ordinateur (CAD) sont fournis au format STL. Il s'agit de maillages triangulaires sans couleurs. La couleur principale de chaque objet est fournie séparément dans un fichier texte. Des maillages reconstruits à partir des images d'entraînement sont aussi fournis, eux avec des couleurs.

5.3.2 Protocole d'acquisition

Voici ce qu'on peut lire sur le protocole d'acquisition des images dans l'article :

“The training and test images were captured with the aid of the setup shown in Fig. 3. It consists of a turntable, where the imaged objects were placed, and a jig with adjustable tilt, to which the sensors were attached. A marker field used for camera pose estimation was affixed to the turntable. The field was extended vertically to the sides of the turntable to facilitate pose estimation at lower elevations. To capture training images, the objects were placed in the middle of the turntable and in the front of a black screen, which ensured a uniform background at all elevations. To introduce a non-uniform background in the test images, a sheet of plywood with markers at its edges was placed on the top of the turntable. In some scenes, the objects were placed on the top of other objects (e.g. books) to give them different elevations and thus invalidate a ground plane assumption that might be made by an evaluated method.”

[...]

“T-LESS offers training images of every object in isolation from a full view sphere. These images were obtained with a systematic acquisition procedure which uniformly sampled elevation from 85° to -85° with a 10° step and the complete azimuth range with a 5° step.”

[...]

“To remove irrelevant parts of the scene in the images periphery, the provided images are cropped versions of the captured ones. Resolution of the provided

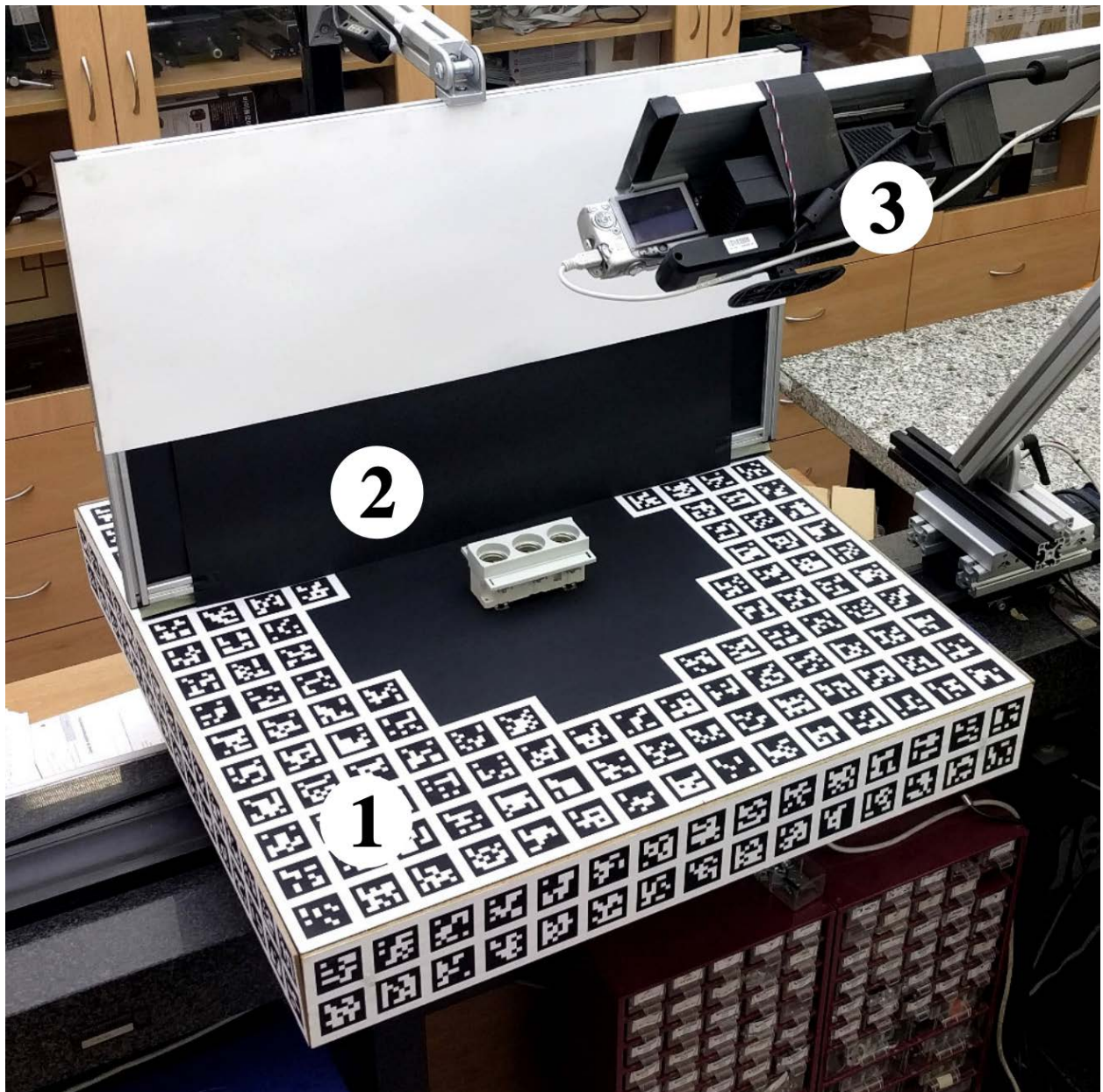


FIGURE 5.12 – Matériel utilisé pour l’acquisition de T-LESS : 1. Plateau tournant avec marqueurs de repères. 2. Fond assurant un arrière-plan noir utilisé seulement pour l’acquisition des images d’entraînement. 3. Les trois caméras sont attachées à un support dont l’angle par rapport à la table est ajustable. Source [56].

	Train	Test
Kinect	400×400 px	720×540 px
Carminé	400×400 px	720×540 px
Canon	1900×1900 px	2560×1920 px

TABLE 5.6 – Résolution des images de T-LESS.

images is as follows : 400×400 px for training RGB-D images from Carminé and Kinect, 1900×1900px for training RGB images from Canon, 720×540px for test RGB-D images from Carminé and Kinect and 2560×1920 px for test RGB images from Canon. ”

[...]

“To obtain ground truth 6D object poses for images of a test scene, a dense 3D model of the scene was first reconstructed with the system of Steinbrücker et al. [44]. This was accomplished using all 504 RGB-D images of the scene along with the sensor poses estimated using the turntable markers. The CAD object models were then manually aligned to the scene model. To increase accuracy, the object models were rendered into several selected high-resolution scene images from Canon, misalignments were identified and the poses were manually refined accordingly. This process was repeated until a satisfactory alignment of the renderings with the scene images was achieved. The final poses were distributed to all test images with the aid of the known camera-to-turntable coordinate transformations.

T-LESS : An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects [9], Section 3.1, Pages 883-884.

Le matériel utilisé est présenté à la figure 5.12 issue de l'article des auteurs [56]. Pour acquérir les images, les objets sont placés sur un plateau tournant couvert de marqueurs. Il y a aussi des marqueurs sur ses côtés pour permettre l'estimation de sa pose, même lorsque la caméra est très basse, et donc voit peu le dessus du plateau. Le plateau tourne avec un pas de 5°. Les trois caméras sont fixées sur un profilé en aluminium dont l'orientation par rapport à la table peut être ajustée manuellement. L'orientation est ajustée par pas de 10° en commençant à 85° et en terminant à 5°.

Pour les images d'entraînement, chaque objet est placé individuellement au centre du plateau devant un fond noir.

Pour les images de test, une plaque de contre-plaqué est placée au centre du plateau, sous les objets. Les objets sont eux-mêmes disposés sur des livres ou dans des récipients (bols, boîtes). Plusieurs objets sont visibles dans ces images et plusieurs instances d'un objet peuvent être présentes à la fois. Leurs poses sont obtenues en reconstruisant un maillage de la scène. Les modèles 3D des objets sont alignés sur ce maillage. Par la suite, les rendus 3D des objets sont affichés par-dessus les images et les alignements des objets sont corrigés manuellement. Cette dernière étape est répétée jusqu'à ce que les résultats soient satisfaisants. Après quoi, la pose des objets est calculée dans chaque autre image grâce au plateau de marqueurs dont la pose par rapport à la caméra est connue.

Les images fournies ne sont pas les photos brutes. Elles sont recadrées pour éliminer la périphérie, jugée *"inintéressante"* par les auteurs. Les images d'entraînement sont aussi masquées de façon à éliminer tout marqueur pouvant apparaître. Plus on s'éloigne de l'objet, plus les pixels sont noircis en suivant un dégradé. Les résolutions obtenues selon les caméras sont détaillées au tableau 5.6 ci-dessous.

5.3.3 Protocole d'utilisation

T-LESS a été produit en même temps que les auteurs développaient leur méthode de détection 3D présentée dans l'article "*Detection and Fine 3D Pose Estimation of Textureless Objects in RGB-D Images*" [55]. Cette méthode, tout comme LINEMOD [51], utilise des patrons qui sont calculés à une certaine distance et orientation autour de l'objet. Chaque patron décrit donc une pose de l'objet. Le problème de détection 3D est alors rapporté à un problème de classification, qui, étant donnée une image de l'objet, cherche à identifier le patron correspondant. Les patrons utilisés par les auteurs sont calculés sur la carte de profondeurs [13] et sur l'image couleur [49], à une distance fixe de l'objet, où celui-ci occupe le "mieux" le patron à calculer. Les patrons sont donc tous calculés à la même distance de l'objet et sont ensuite redimensionnés pour tenir compte de la distance à l'objet. Cela explique pourquoi les auteurs ont photographié les images d'entraînement de T-LESS à une distance fixe des objets, et avec les objets centrés dans l'image. T-LESS est conçu spécialement pour ce type de méthodes et non pour l'apprentissage profond.

Le problème est que les réseaux de neurones ne peuvent apprendre à modéliser les distances aux objets que si ces objets sont photographiés à plusieurs distances dans les images d'entraînement. Les variations d'échelles dues aux changements de distances doivent être présente dans les images d'entraînements. Or, avec T-LESS, l'inverse risque de se produire. On va spécialiser le réseau à une seule échelle, une distance à l'objet. Mais des changements de distances, et donc d'échelles, sont présents dans les images de tests. Un écart existe donc entre images d'entraînements et de tests. Cet écart est appelé "écart sémantique", ou "vide sémantique" car la tâche demandée sur les images de tests est plus complexe que celle présentée dans les images d'entraînement.

Les méthodes de détection 3D basées sur l'apprentissage profond qui ont présenté des résultats sur T-LESS utilisent les images d'entraînement et appliquent des augmentations pour essayer de combler le vide sémantique. Ainsi, BB8 [91] redimensionne les images d'un facteur compris entre $0.8 \times$ et $1.2 \times$ pour tenir compte du changement de distance à l'objet.

Plus récemment, les données d'entraînement synthétiques se sont imposées sur ce jeu de données. Pour réaliser ces rendus plusieurs logiciels ont été utilisés comme Blender, Unity 3D ou l'Unreal Engine. Les objets sont placés aléatoirement dans des scènes 3D reproduisant des environnements réels comme des cuisines, salon, paysage, etc. Des sources de lumière sont aussi placées aléatoirement dans la scène pour reproduire les conditions d'éclairage à différent moment de la journée, à différentes saisons, ou bien des conditions d'éclairages en intérieurs. Les rendus obtenus sont utilisés pour entraîner le réseau au lieu des images réelles. Cette technique est devenue possible grâce aux progrès réalisés en photoréalisme ces dernières années. Si les rendus ne sont pas aussi vrais que nature, on s'expose à nouveau à un écart sémantique entre images d'entraînements et de tests, non pas du à la position des objets dans les images, mais à leurs apparences.

5.3.4 Intérêts et points forts

Le premier point fort de T-LESS sont ses modèles 3D CAD de grande précision. Grâce à eux, il n'y a pas de problème de normale ou de triangle manquant, ce qui permet de calculer des rendus 3D de qualité. Il en va de même pour toutes les étiquettes calculées à partir des modèles 3D.

Les auteurs ont été les premiers à proposer de calculer des mesures d'erreurs entre cartes de profondeur rendues et réelles pour évaluer la qualité des étiquettes de pose. Les étiquettes de T-LESS sont très précises. Pour la caméra Carmin Primesense, l'erreur moyenne est $-0.6mm \pm 8.12mm$ et pour la Kinect elle est $4.46mm \pm 11.76mm$.

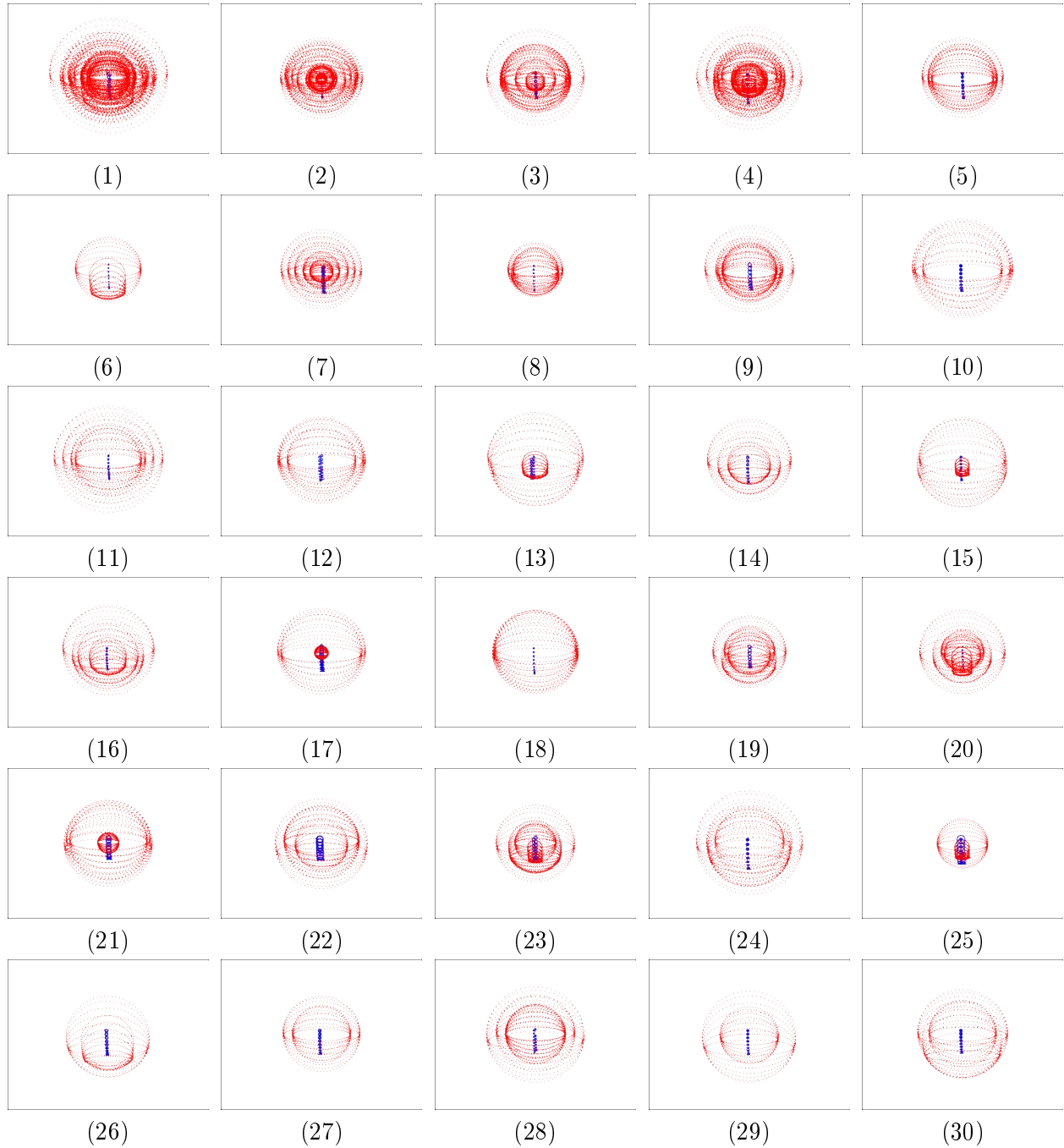


FIGURE 5.13 – Positions où apparaissent les objets dans les images d’entraînement (en bleu) et de test (en rouge) de T-LESS.

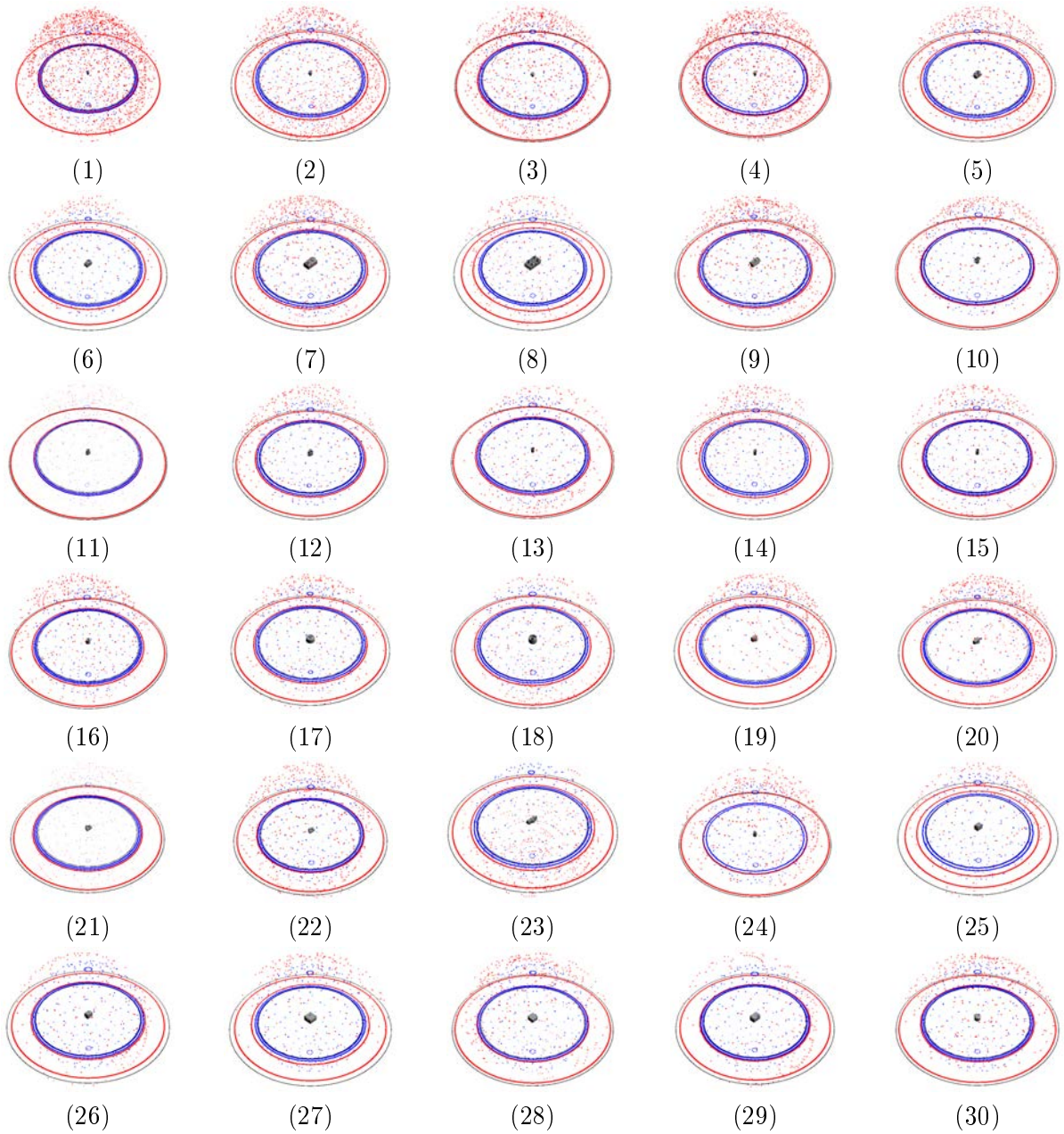


FIGURE 5.14 – Positions où se trouvait la caméra lorsqu'elle a photographié les objets dans les images d'entraînement (en bleu) et de test (en rouge) de T-LESS.

Objet	$\min(v)$	$\max(v)$	$\mu(v) \pm \sigma(v)$
00	00.00 %	100.00 %	73.97 \pm 32.96 %
01	00.00 %	100.00 %	71.56 \pm 31.66 %
02	00.00 %	100.00 %	87.53 \pm 21.60 %
03	00.00 %	100.00 %	80.01 \pm 27.76 %
04	00.00 %	100.00 %	87.29 \pm 24.53 %
05	21.36 %	100.00 %	96.28 \pm 12.44 %
06	27.10 %	100.00 %	89.00 \pm 15.91 %
07	32.15 %	100.00 %	88.31 \pm 14.51 %
08	07.06 %	100.00 %	93.76 \pm 14.55 %
09	00.52 %	100.00 %	93.23 \pm 17.68 %
10	00.00 %	100.00 %	84.61 \pm 28.47 %
11	00.00 %	100.00 %	87.84 \pm 24.19 %
12	00.00 %	100.00 %	86.81 \pm 23.75 %
13	00.87 %	100.00 %	83.00 \pm 22.83 %
14	04.31 %	100.00 %	88.68 \pm 18.51 %
15	00.00 %	100.00 %	86.67 \pm 21.98 %
16	07.03 %	100.00 %	88.76 \pm 17.99 %
17	00.00 %	100.00 %	88.71 \pm 21.31 %
18	00.00 %	100.00 %	86.57 \pm 24.14 %
19	00.00 %	100.00 %	77.95 \pm 25.52 %
20	00.00 %	100.00 %	73.11 \pm 33.77 %
21	00.00 %	100.00 %	78.25 \pm 27.44 %
22	07.13 %	100.00 %	90.16 \pm 17.49 %
23	00.05 %	100.00 %	87.39 \pm 22.88 %
24	01.95 %	100.00 %	87.05 \pm 21.59 %
25	17.30 %	100.00 %	94.31 \pm 15.11 %
26	00.35 %	096.68 %	65.77 \pm 18.70 %
27	00.00 %	100.00 %	79.44 \pm 26.42 %
28	09.36 %	100.00 %	93.94 \pm 14.87 %
29	00.00 %	100.00 %	91.32 \pm 19.12 %
Tous	00.00 %	100.00 %	83.06 \pm 26.09 %

TABLE 5.7 – Visibilités minimums, maximums, moyennes et écarts types des objets dans les images de test du jeu de données T-LESS.

$\min(d)/\min_{train}(d)$	$\max(d)/\max_{train}(d)$	$\min(a)/\min_{train}(a)$	$\max(a)/\max_{train}(a)$
1.031	1.422	62.78 %	113.63 %
1.094	1.363	55.40 %	102.50 %
1.070	1.407	58.28 %	96.86 %
1.077	1.413	51.02 %	120.49 %
1.097	1.367	64.14 %	83.33 %
1.096	1.339	65.56 %	83.94 %
1.091	1.356	84.26 %	71.81 %
1.150	1.317	64.32 %	66.58 %
1.079	1.367	69.53 %	74.07 %
1.062	1.409	69.60 %	82.19 %
1.030	1.411	61.47 %	93.21 %
1.080	1.375	57.42 %	88.04 %
1.069	1.414	65.23 %	106.61 %
1.098	1.382	71.21 %	116.86 %
1.062	1.392	64.85 %	89.36 %
1.070	1.407	56.87 %	82.42 %
1.080	1.364	61.32 %	79.67 %
1.091	1.392	62.83 %	78.63 %
1.089	1.350	58.14 %	81.35 %
1.070	1.367	64.26 %	82.43 %
1.067	1.375	57.44 %	77.84 %
1.058	1.373	58.95 %	79.18 %
1.102	1.356	72.21 %	80.83 %
1.066	1.411	58.44 %	118.82 %
1.162	1.306	70.58 %	71.94 %
1.071	1.359	71.00 %	81.57 %
1.108	1.357	65.79 %	75.81 %
1.047	1.387	64.90 %	79.94 %
1.061	1.373	74.73 %	82.66 %
1.060	1.394	60.05 %	88.19 %
1.030	1.422	62.78 %	66.58 %

 TABLE 5.8 – Statistiques d'échelles et d'aires des objets dans les images de T-LESS. d est la distance séparant l'objet et la caméra. Plus l'objet est loin plus il apparaît petit à l'image et inversement. a est l'air de l'objet.

Objet	$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
00	04.21 \pm 12.42 mm	08.69 \pm 09.83 mm	7.69 %
01	05.41 \pm 11.90 mm	08.67 \pm 09.78 mm	6.33 %
02	04.97 \pm 10.57 mm	07.50 \pm 08.95 mm	5.00 %
03	05.29 \pm 09.85 mm	07.26 \pm 08.50 mm	5.73 %
04	03.38 \pm 09.42 mm	06.15 \pm 07.89 mm	4.16 %
05	02.17 \pm 08.33 mm	05.34 \pm 06.75 mm	2.56 %
06	04.39 \pm 10.12 mm	07.51 \pm 08.07 mm	4.42 %
07	08.33 \pm 12.40 mm	10.98 \pm 10.13 mm	3.25 %
08	04.37 \pm 09.36 mm	06.82 \pm 07.76 mm	3.70 %
09	05.28 \pm 10.68 mm	07.94 \pm 08.89 mm	5.57 %
10	01.86 \pm 10.54 mm	06.76 \pm 08.29 mm	6.00 %
11	01.86 \pm 10.54 mm	06.73 \pm 08.32 mm	4.75 %
12	03.84 \pm 10.77 mm	07.49 \pm 08.64 mm	4.89 %
13	03.99 \pm 10.68 mm	07.19 \pm 08.85 mm	3.09 %
14	04.42 \pm 10.50 mm	07.39 \pm 08.66 mm	4.28 %
15	03.95 \pm 10.45 mm	07.17 \pm 08.57 mm	3.80 %
16	02.44 \pm 10.56 mm	06.69 \pm 08.53 mm	4.98 %
17	11.08 \pm 09.65 mm	11.84 \pm 08.70 mm	4.18 %
18	10.23 \pm 10.29 mm	11.00 \pm 09.47 mm	6.48 %
19	10.36 \pm 10.72 mm	11.46 \pm 09.53 mm	6.15 %
20	10.07 \pm 10.66 mm	11.26 \pm 09.39 mm	5.56 %
21	12.15 \pm 10.90 mm	13.23 \pm 09.54 mm	7.81 %
22	07.48 \pm 10.27 mm	08.62 \pm 09.34 mm	5.12 %
23	04.74 \pm 10.37 mm	07.25 \pm 08.80 mm	6.63 %
24	02.77 \pm 08.81 mm	05.32 \pm 07.54 mm	3.75 %
25	05.23 \pm 09.69 mm	07.38 \pm 08.18 mm	4.25 %
26	01.59 \pm 08.98 mm	05.53 \pm 07.25 mm	5.16 %
27	03.17 \pm 09.36 mm	05.88 \pm 07.94 mm	4.92 %
28	02.61 \pm 07.94 mm	04.94 \pm 06.74 mm	4.23 %
29	02.93 \pm 08.06 mm	04.93 \pm 07.02 mm	5.08 %
Tous	5.36 \pm 10.65 mm	07.97 \pm 08.87 mm	5.46 %

TABLE 5.9 – Erreurs mesurées dans les cartes de profondeur d'entraînement du jeu de données T-LESS.

$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
00.07 \pm 10.47 mm	07.38 \pm 07.43 mm	05.47 %
00.93 \pm 10.97 mm	07.75 \pm 07.82 mm	06.38 %
00.77 \pm 11.11 mm	07.85 \pm 07.90 mm	06.13 %
01.41 \pm 10.34 mm	07.13 \pm 07.62 mm	09.93 %
-0.49 \pm 09.84 mm	07.22 \pm 06.71 mm	02.19 %
-0.00 \pm 09.59 mm	07.04 \pm 06.51 mm	01.69 %
07.80 \pm 14.29 mm	12.51 \pm 10.42 mm	04.60 %
08.56 \pm 15.60 mm	13.73 \pm 11.31 mm	04.00 %
02.08 \pm 10.86 mm	07.62 \pm 08.02 mm	02.95 %
00.33 \pm 11.35 mm	07.93 \pm 08.13 mm	02.82 %
-0.61 \pm 10.52 mm	07.51 \pm 07.39 mm	02.36 %
-0.01 \pm 10.96 mm	07.68 \pm 07.82 mm	02.39 %
-2.69 \pm 09.99 mm	08.00 \pm 06.55 mm	02.55 %
-0.21 \pm 11.57 mm	08.34 \pm 08.02 mm	05.01 %
-0.64 \pm 11.03 mm	08.08 \pm 07.54 mm	03.38 %
-0.79 \pm 10.10 mm	07.38 \pm 06.94 mm	02.56 %
-0.42 \pm 09.79 mm	06.99 \pm 06.86 mm	01.57 %
08.80 \pm 11.50 mm	10.90 \pm 09.54 mm	26.41 %
04.60 \pm 10.44 mm	07.81 \pm 08.32 mm	06.26 %
03.99 \pm 11.01 mm	07.98 \pm 08.57 mm	05.90 %
06.06 \pm 10.41 mm	08.53 \pm 08.50 mm	12.94 %
07.06 \pm 11.00 mm	09.25 \pm 09.24 mm	13.77 %
04.22 \pm 11.36 mm	08.40 \pm 08.74 mm	15.57 %
-1.16 \pm 09.12 mm	06.92 \pm 06.06 mm	21.15 %
01.30 \pm 10.04 mm	07.28 \pm 07.03 mm	24.67 %
03.39 \pm 10.83 mm	08.02 \pm 08.02 mm	28.20 %
00.98 \pm 10.60 mm	07.83 \pm 07.21 mm	02.13 %
01.51 \pm 10.52 mm	07.87 \pm 07.15 mm	02.61 %
00.85 \pm 10.48 mm	07.73 \pm 07.18 mm	02.42 %
01.49 \pm 10.35 mm	07.47 \pm 07.32 mm	02.48 %
03.02 \pm 12.15 mm	08.97 \pm 08.74 mm	07.74 %

TABLE 5.10 – Erreurs mesurées dans les cartes de profondeur de test du jeu de données T-LESS.

T-LESS propose plusieurs facteurs de difficultés. Tout d'abord, les objets n'ont pas de couleurs ni de textures discriminantes et certains se ressemblent. Par exemple, les objets 1 et 2, 5 et 6 ou encore 17 et 18 (voir la figure 5.10). D'autres sont des sous-parties comme les objets 6 et 7. La plupart présentent des symétries ou des répétitions de formes modulo un angle qui créent des ambiguïtés de pose.

T-LESS présente de fortes occlusions.

Il y a aussi des changements importants de luminosités et de couleurs entre les images d'entraînement et de test. Ces changements sont dus à la présence du fond noir dans les images d'entraînement. En effet, ce dernier bloque la lumière venant de derrière l'objet. De plus, les caméras numériques sont sujettes à des changements de focus, de couleur et même de luminosité selon l'ouverture de l'iris et le temps d'aperture.

Tous ces facteurs font de T-LESS un jeu de données intéressant et difficile qui permet réellement d'évaluer la robustesse d'une méthode.

5.3.5 Limitations

Premièrement, les modèles 3D sont très détaillés et font plusieurs mégaoctets. Ce qui était un avantage pour le calcul des rendus 3D et des étiquettes, est un désavantage majeur pour le calcul de toute fonction de coût ou mesure de performance utilisant les modèles 3D. Si les modèles avaient été paramétriques, alors il aurait été possible de faire des exports avec moins de polygones. On pourrait même envisager de calculer des mesures directement sur les paramètres du modèle. Au lieu de cela, les auteurs ont dû décimer les modèles avec MeshLab selon la technique du point à mi-distance (3 itérations, seuil des arêtes fixé à $2mm$). Cette technique réduit peu le poids des modèles qui est toujours de l'ordre du mégaoctet.

Deuxièmement, les symétries sont annotées (dans la version BOP), mais les sous-parties d'objets ne le sont pas. Il existe des représentations de ces étiquettes pour la classification qui permettent d'affecter plusieurs catégories et sous-catégories tenant compte de cette imbrication comme WordNet [80]. De même, pour la pose, il est possible de définir la relation entre plusieurs objets comme une hiérarchie. Ce faisant, on obtient toutes les relations mathématiques permettant de passer de la pose d'un objet à l'autre. Les différents objets peuvent donc contribuer à l'estimation de la pose du groupe. Malheureusement, ce n'est pas possible sans plus de travail dans T-LESS.

Dernier point, les images d'entraînement sont photographiées à une distance fixe des objets. Les images de tests de T-LESS sont photographiés à des distances plus éloignées des objets. En moyenne les distances aux objets sont de l'ordre de 1.06 à 1.40 fois plus grandes. En termes de taille à l'image, les objets apparaissent donc souvent plus petit dans les images de tests. Nous avons mesuré qu'ils apparaissent entre 62.78 % plus petit et 120.49 % plus grand. Ces statistiques varient d'un objet à l'autre et sont présentées au tableau 5.8.

Comme mentionné ci-dessus, ces images sont adaptées à des méthodes basées sur des patrons de poses. Mais ces images ne couvrent pas tout l'espace des poses de tests comme on peut le voir à la figure 5.14. Les images d'entraînement ne présentent aucune occlusion, car l'objet est seul, et celles de test en contiennent beaucoup avec de nombreux objets annotés. De ce fait, il est très difficile pour une méthode qui apprend seulement à partir des images d'entraînement de généraliser sur les images de tests. On appelle ce problème le vide ou gouffre sémantique. Pour combler le vide, l'augmentation de données apporte des améliorations, mais ne le résout pas complètement.



FIGURE 5.15 – Objets de la base d'objets YCB-OBJECTS [14, 16, 15] utilisés dans le jeu de données YCB-VIDÉO [126].

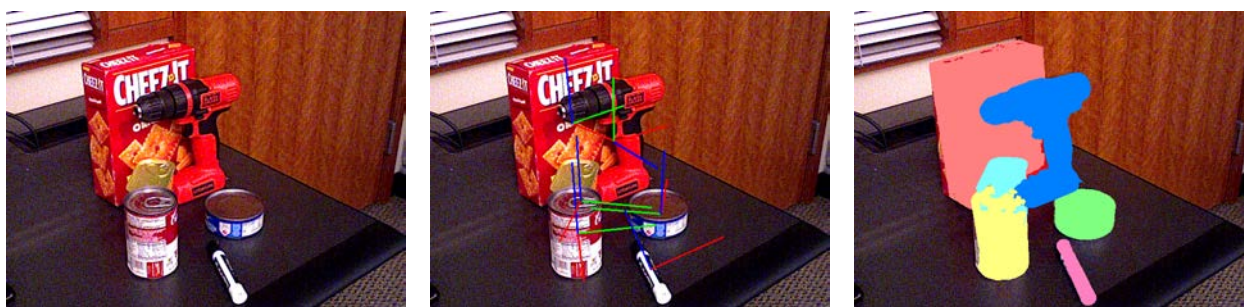


FIGURE 5.16 – Exemple d'une image du jeu de données YCB-VIDÉO [126]. La pose de chaque objet est annoté comme on peut le voir dans la deuxième image. Les masque de segmentation tenant compte des occlusions sont aussi fournis comme le montre la troisième image.

5.4 YCB-VIDEO

Le jeu de données a été produit par Xiang Yu, Schmidt Tanner, Narayanan Venkatraman et Fox Dieter lors du développement de leur méthode de détection 3D nommée Pose CNN [126]. Ils ont nommé ce jeu de données "YCB-Vidéo".

5.4.1 Informations générales

Les objets qu'ils ont utilisés sont ceux de Yale-CMU-Berkeley (YCB). YCB est un répertoire d'objets et leurs modèles 3D, conçu par Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, Aaron Dollar [14, 16, 15]. Ces objets sont disponibles, de sorte que d'autres chercheurs puissent proposer des protocoles et des analyses comparatives ("benchmarks") dans le domaine de la préhension et de la manipulation robotique. Les objets sont présentés à la figure 5.15 et sur le site du projet <https://www.ycbbenchmarks.com/> ainsi que dans les publications scientifiques des auteurs. Il s'agit d'objets du quotidien qui présentent différentes formes, textures et tailles. Ils sont regroupés en cinq groupes :

- **"Food"** : Boîtes, conserves, bouteilles, fruits.
- **"Kitchen"** : Sautreuse, ustensiles, vaisselle, bouteilles, nappe.
- **"Tool"** : Perceuse, clés, tourne-vis, ciseaux, vis, boulons, pinces, tasseau.
- **"Shape"** : Balles, billes, dés, rondelles, chaîne, gobelets, ficelle.
- **"Task"** : Boite, LEGO, jeu de plateau, calculette, Rubix cub, T-shirt, avion.

Il est possible d'acheter les objets ou un ensemble individuel en passant par le site.

Pour créer le jeu de données YVB-Vidéo, les auteurs de PoseCNN ont utilisé 21 objets de YCB. Ils ont filmés 92 vidéos avec une caméra "Asus Xtion Pro Live RGB-D" pour un

total de 133827 images. Les vidéos sont fournies sous forme d'images, où pour chaque image, la pose des objets présents est annotée. Un exemple est présenté à la figure 5.16.

5.4.2 Protocole d'acquisition

Voici ce que l'on peut lire dans l'article [126] au sujet de la création du jeu de données :

“Object-centric datasets providing ground-truth annotations for object poses and/or segmentations are limited in size by the fact that the annotations are typically provided manually. For example, the popular LINEMOD dataset [50] provides manual annotations for around 1,000 images for each of the 15 objects in the dataset. While such a dataset is useful for evaluation of model-based pose estimation techniques, it is orders of magnitude smaller than a typical dataset for training state-of-the-art deep neural networks. One solution to this problem is to augment the data with synthetic images. However, care must be taken to ensure that performance generalizes between real and rendered scenes.”

[...]

“To avoid annotating all the video frames manually, we manually specify the poses of the objects only in the first frame of each video. Using Signed Distance Function (SDF) representations of each object, we refine the pose of each object in the first depth frame. Next, the camera trajectory is initialized by fixing the object poses relative to one another and tracking the object configuration through the depth video. Finally, the camera trajectory and relative object poses are refined in a global optimization step.”

[...]

“Asus Xtion Pro Live RGB-D camera in fast-cropping mode, which provides RGB images at a resolution of 640×480 at 30 FPS”

[...]

“The full dataset comprises 133,827 images, two full orders of magnitude larger than the LINEMOD dataset”

[...]

“Note that our annotation accuracy suffers from several sources of error, including the rolling shutter of the RGB sensor, inaccuracies in the object models, slight asynchrony between RGB and depth sensors, and uncertainties in the intrinsic and extrinsic parameters of the cameras.”

PoseCNN : A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes [126]. Page 4, Partie IV.

La principale motivation des auteurs était donc d'avoir de nombreuses images pour pouvoir entraîner des réseaux de neurones profonds. La deuxième contrainte était d'annoter le moins possible les images manuellement. Il est vrai que les annotations manuelles sont sources d'erreurs, surtout pour la détection 3D où l'annotation manuelle dans les images est très fastidieuse.

Le protocole mis en place pour répondre à ces contraintes est le suivant. Plusieurs objets sont disposés sur une table. La caméra est tenue en main et est déplacée devant la table pendant l'enregistrement d'une séquence. La pose des objets est annotée manuellement dans la première image de la séquence. Grâce à une fonction de distance signée [85] le recalage des modèles est affiné en minimisant la distance entre carte de profondeur rendue et carte de profondeur photographiée. La pose des objets est propagée dans les images suivantes en suivant le déplacement de la caméra. Puis elle est corrigée pour chaque image. Pour finir,

Objet	$\min(v)$	$\max(v)$	$\mu(v) \pm \sigma(v)$	hors-champ
Banana	00.00 %	100 %	88.47 \pm 18.89 %	0.2291 %
Bleach	00.00 %	100 %	81.47 \pm 22.14 %	0.5181 %
Chocolate	00.00 %	100 %	83.89 \pm 22.15 %	0.4389 %
Coffee Can	00.00 %	100 %	82.09 \pm 20.52 %	0.4874 %
Cracker Box	27.43 %	100 %	82.76 \pm 14.41 %	0.0000 %
Foam Brick	00.24 %	100 %	88.88 \pm 18.24 %	0.0000 %
Gelatin Box	08.87 %	100 %	84.91 \pm 22.52 %	0.0000 %
Marker	00.00 %	100 %	85.01 \pm 22.27 %	0.7909 %
Metal Bowl	00.00 %	100 %	74.97 \pm 25.74 %	0.1577 %
Metal Mug	00.00 %	100 %	92.59 \pm 15.72 %	0.9322 %
Mustard	00.00 %	100 %	88.91 \pm 18.89 %	1.5048 %
Pitcher	00.62 %	100 %	93.51 \pm 11.96 %	0.0000 %
Potted Meat	00.00 %	100 %	91.98 \pm 17.40 %	1.1830 %
Power Drill	00.74 %	100 %	87.11 \pm 17.03 %	0.0000 %
Scissors	00.00 %	100 %	88.37 \pm 18.14 %	1.0089 %
Clamp (large)	00.00 %	100 %	88.48 \pm 14.68 %	0.0186 %
Clamp (small)	00.00 %	100 %	91.43 \pm 17.59 %	0.5130 %
Sugar box	00.00 %	100 %	81.40 \pm 22.02 %	1.0698 %
Tomato soup	00.00 %	100 %	92.60 \pm 16.16 %	0.2989 %
Tuna can	00.00 %	100 %	78.78 \pm 24.05 %	0.7964 %
Wood block	08.00 %	100 %	71.87 \pm 18.79 %	0.0000 %
Tous	00.00 %	100 %	86.17 \pm 19.94 %	0.4896 %

TABLE 5.11 – Visibilités minimums, maximums, moyennes et écarts types des objets dans les images d’entraînement du jeu de données YCB-VIDÉO [126].

$\min(v)$	$\max(v)$	$\mu(v) \pm \sigma(v)$
59.80 %	100.00 %	94.25 \pm 10.12%
53.57 %	100.00 %	87.69 \pm 12.52%
52.18 %	076.86 %	64.88 \pm 07.36%
64.79 %	099.96 %	89.93 \pm 09.25%
21.48 %	099.33 %	70.91 \pm 17.98%
31.53 %	099.99 %	87.50 \pm 14.50%
71.70 %	100.00 %	99.38 \pm 02.68%
79.37 %	100.00 %	98.96 \pm 03.09%
45.24 %	099.97 %	75.74 \pm 14.36%
88.80 %	100.00 %	99.15 \pm 01.13%
95.46 %	100.00 %	99.75 \pm 00.34%
87.86 %	099.99 %	97.42 \pm 03.07%
32.29 %	100.00 %	74.32 \pm 23.68%
62.41 %	100.00 %	95.51 \pm 05.89%
53.45 %	083.69 %	68.80 \pm 08.65%
41.66 %	100.00 %	89.58 \pm 13.96%
59.51 %	100.00 %	93.72 \pm 08.42%
61.04 %	100.00 %	90.40 \pm 13.01%
07.54 %	100.00 %	91.13 \pm 19.73%
96.11 %	100.00 %	99.81 \pm 00.33%
61.18 %	080.15 %	72.64 \pm 04.37%
07.54 %	100.00 %	89.19 \pm 15.65%

TABLE 5.12 – Visibilités minimums, maximums, moyennes et écarts types des objets dans les images de test du jeu de données YCV-VIDÉO [126].

la pose des objets et le parcours de la caméra sont affinés lors d’une étape d’optimisation globale.

Les auteurs ont noté plusieurs sources d’erreur dans leur protocole telles que l’effet obturateur déroulant dans les images couleur, les imprécisions dans les modèles 3D qui donnent lieu à un mauvais alignement avec la carte de profondeur, le recalage entre images couleur et cartes de profondeur, et les problèmes d’incertitude des paramètres intrinsèques de la caméra.

5.4.3 Protocole d’utilisation

“In our YCB-Video dataset, we use 80 videos for training, and test on 2949 key frames extracted from the rest 12 test videos.”

PoseCNN : A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes [126]. Page 4, Partie V.

Très simplement quatre-vingt vidéos sont utilisées pour l’entraînement. 2949 images des douze autres vidéos sont utilisées pour les tests.

5.4.4 Intérêts et points forts

Premièrement, les objets de YCB sont soigneusement choisis et scannés. Ils sont issus d’activités du quotidien, ce qui permet de les utiliser dans de nombreux cadres applicatifs. Ils présentent des facteurs de difficulté variés. Par exemple, certains sont peu texturés, d’autres ont des symétries. Le fait qu’ils soient disponibles à l’achat est aussi intéressant, car cela permet à d’autres chercheurs de vérifier eux-mêmes les résultats, ou de contribuer à de nouvelles séquences pour le jeu de données.

Deuxièmement, les auteurs ont produit un nombre impressionnant de séquences, ce qui, à terme, représente un nombre d’autant plus important d’images. Ils ont parfaitement atteint leurs objectifs pour ce qui est d’avoir suffisamment d’images d’entraînement pour des méthodes utilisant des réseaux de neurones profonds.

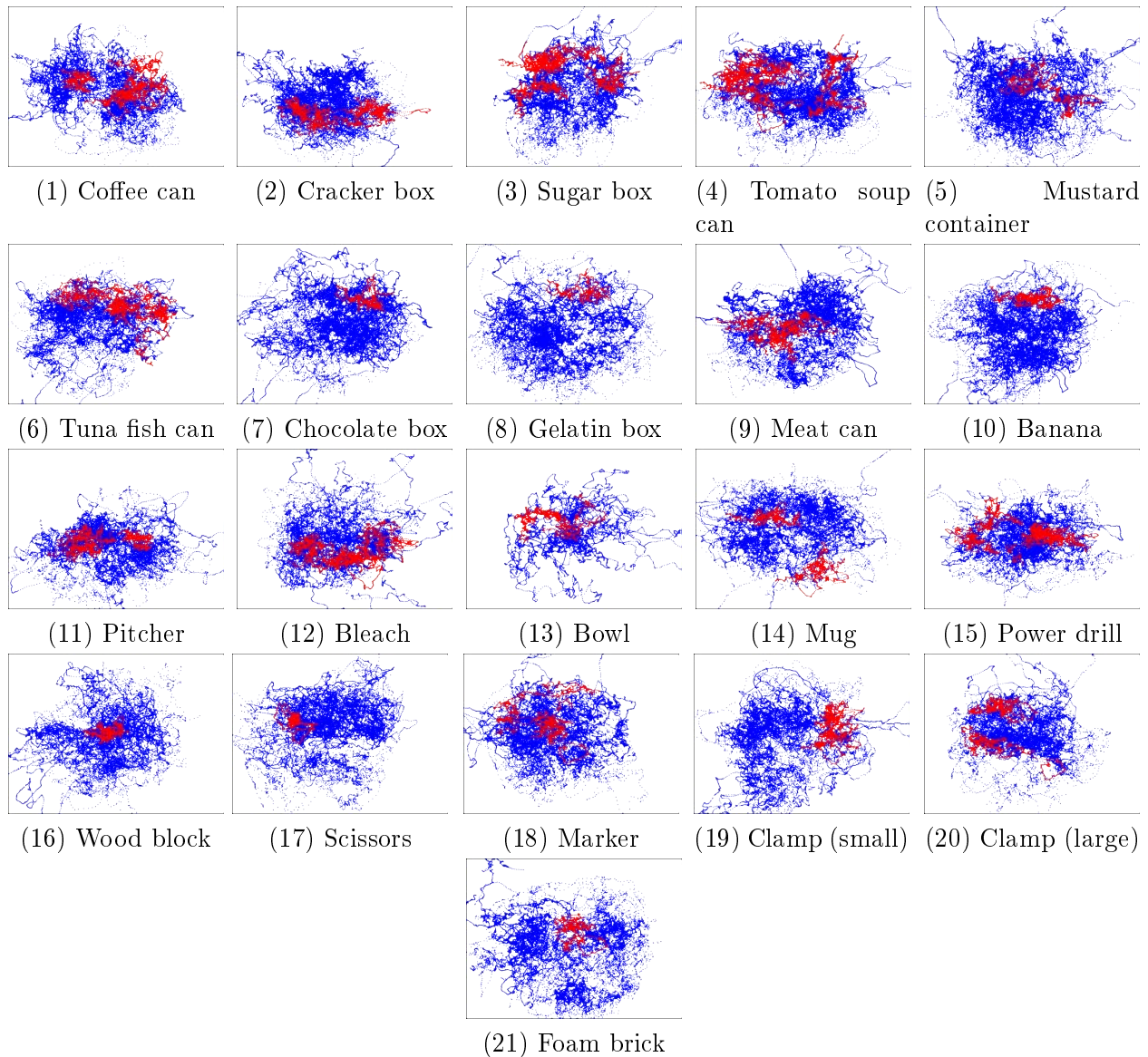


FIGURE 5.17 – Positions où apparaissent les objets dans les images d’entraînement (en bleu) et de test (en rouge) de YCB-VIDÉO.

Troisièmement, les images présentent une bonne proportion d’objets occultés comme le montre les tableaux 5.11 et 5.12. On pourra donc entraîner le réseau de neurones à faire face à ce facteur de difficulté.

5.4.5 Limitations

Contrairement à LINEMOD ou T-LESS la caméra capture ici une vidéo. Le problème d’obturateur déroulant aurait pu être évité si les auteurs avaient photographié les objets au lieu de les filmer. Bien évidemment, photographier les objets sans un déclencheur automatisé aurait été plus long et aurait fourni moins d’images. Ou alors, les images auraient été photographiées moins proches les unes des autres. Ce qui aurait pu pénaliser la méthode d’annotation automatique. Néanmoins, la même méthode de suivi actif avait déjà été utilisée avec succès dans LINEMOD+OCCLUDED sur des images éparses.

La caméra est tenue en main par la personne filmant les images. Ainsi, les images filmées sont réparties le long du chemin suivi par la caméra, comme le montre la figure 5.18 qui

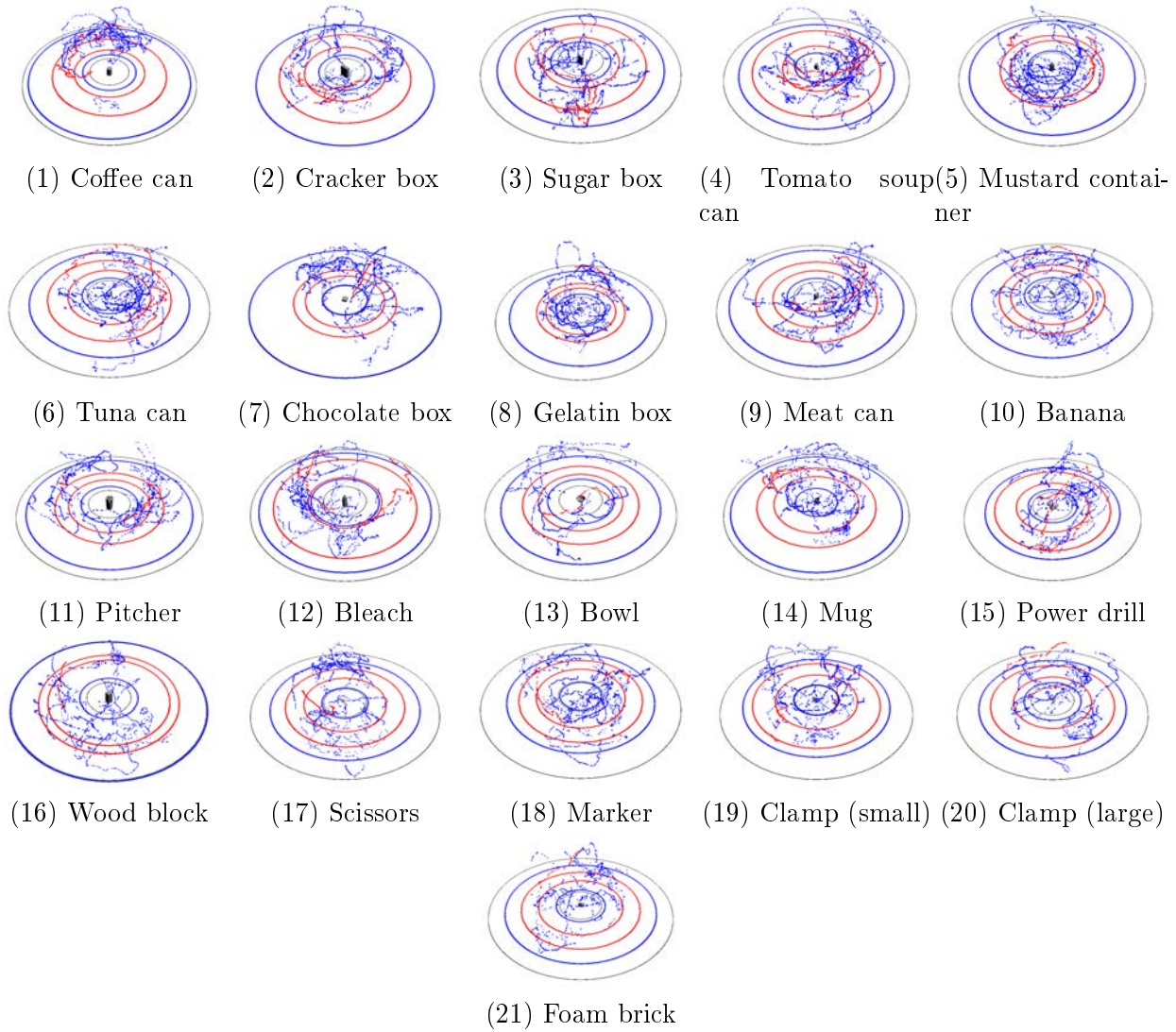


FIGURE 5.18 – Positions où se trouvait la caméra lorsqu'elle a photographié les objets dans les images d'entraînement (en bleu) et de tests (en rouge) de YCV-VIDÉO [126].

Objet	$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
Banana	$1.25 \pm 5.39 \text{ mm}$	$2.45 \pm 4.96 \text{ mm}$	22.12 %
Bleach	$0.65 \pm 3.61 \text{ mm}$	$2.27 \pm 2.88 \text{ mm}$	11.19 %
Chocolate	$1.10 \pm 4.92 \text{ mm}$	$2.72 \pm 4.25 \text{ mm}$	14.88 %
Coffee Can	$0.26 \pm 3.94 \text{ mm}$	$2.41 \pm 3.13 \text{ mm}$	20.08 %
Cracker Box	$0.69 \pm 4.64 \text{ mm}$	$3.00 \pm 3.60 \text{ mm}$	10.33 %
Foam Brick	$0.60 \pm 3.96 \text{ mm}$	$2.38 \pm 3.22 \text{ mm}$	13.47 %
Gelatin Box	$1.34 \pm 4.90 \text{ mm}$	$2.49 \pm 4.43 \text{ mm}$	20.58 %
Marker	$4.11 \pm 9.39 \text{ mm}$	$5.24 \pm 8.81 \text{ mm}$	36.19 %
Metal Bowl	$1.81 \pm 6.89 \text{ mm}$	$4.06 \pm 5.85 \text{ mm}$	28.86 %
Metal Mug	$1.76 \pm 5.41 \text{ mm}$	$2.98 \pm 4.84 \text{ mm}$	16.71 %
Mustard	$0.55 \pm 3.67 \text{ mm}$	$2.10 \pm 3.06 \text{ mm}$	12.05 %
Pitcher	$1.03 \pm 5.45 \text{ mm}$	$2.89 \pm 4.73 \text{ mm}$	12.90 %
Potted Meat	$1.20 \pm 4.04 \text{ mm}$	$2.42 \pm 3.44 \text{ mm}$	21.12 %
Power Drill	$1.35 \pm 6.05 \text{ mm}$	$3.24 \pm 5.29 \text{ mm}$	27.86 %
Scissors	$3.32 \pm 6.81 \text{ mm}$	$4.19 \pm 6.31 \text{ mm}$	15.45 %
Clamp (large)	$3.27 \pm 8.12 \text{ mm}$	$4.41 \pm 7.56 \text{ mm}$	35.36 %
Clamp (small)	$2.62 \pm 6.62 \text{ mm}$	$3.71 \pm 6.08 \text{ mm}$	30.45 %
Sugar Box	$0.65 \pm 3.80 \text{ mm}$	$2.25 \pm 3.13 \text{ mm}$	15.68 %
Tomato Soup	$0.70 \pm 4.33 \text{ mm}$	$2.73 \pm 3.43 \text{ mm}$	20.94 %
Tuna Fish Can	$1.16 \pm 5.46 \text{ mm}$	$2.56 \pm 4.96 \text{ mm}$	41.98 %
Wood Block	$0.50 \pm 4.05 \text{ mm}$	$2.37 \pm 3.32 \text{ mm}$	13.59 %
Tous	$1.11 \pm 5.09 \text{ mm}$	$2.80 \pm 4.39 \text{ mm}$	20.60 %

TABLE 5.13 – Erreurs mesurées dans les cartes de profondeur d'entraînement du jeu de données YCB-VIDÉO [126].

$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
$+1.19 \pm 4.77 \text{ mm}$	$2.29 \pm 4.35 \text{ mm}$	14.34 %
$+0.35 \pm 3.35 \text{ mm}$	$2.14 \pm 2.61 \text{ mm}$	10.47 %
$+0.96 \pm 3.12 \text{ mm}$	$2.03 \pm 2.56 \text{ mm}$	05.15 %
$-0.18 \pm 4.79 \text{ mm}$	$3.11 \pm 3.66 \text{ mm}$	19.22 %
$+0.52 \pm 5.12 \text{ mm}$	$3.15 \pm 4.07 \text{ mm}$	16.83 %
$+0.30 \pm 4.08 \text{ mm}$	$2.64 \pm 3.12 \text{ mm}$	13.56 %
$+0.87 \pm 2.31 \text{ mm}$	$1.59 \pm 1.89 \text{ mm}$	02.76 %
$+1.88 \pm 5.83 \text{ mm}$	$2.99 \pm 5.35 \text{ mm}$	23.40 %
$+1.23 \pm 4.19 \text{ mm}$	$2.13 \pm 3.82 \text{ mm}$	08.68 %
$+1.64 \pm 4.47 \text{ mm}$	$3.15 \pm 3.57 \text{ mm}$	15.11 %
$+0.63 \pm 3.15 \text{ mm}$	$1.92 \pm 2.58 \text{ mm}$	07.27 %
$+0.88 \pm 4.07 \text{ mm}$	$2.18 \pm 3.55 \text{ mm}$	10.39 %
$+1.27 \pm 5.09 \text{ mm}$	$2.93 \pm 4.35 \text{ mm}$	36.90 %
$+0.82 \pm 5.22 \text{ mm}$	$2.79 \pm 4.48 \text{ mm}$	21.51 %
$+4.16 \pm 8.55 \text{ mm}$	$5.55 \pm 7.72 \text{ mm}$	18.08 %
$+2.03 \pm 5.91 \text{ mm}$	$3.49 \pm 5.18 \text{ mm}$	42.40 %
$+3.12 \pm 6.83 \text{ mm}$	$4.64 \pm 5.90 \text{ mm}$	36.42 %
$+1.27 \pm 3.98 \text{ mm}$	$2.32 \pm 3.47 \text{ mm}$	06.66 %
$+2.38 \pm 4.55 \text{ mm}$	$3.61 \pm 3.65 \text{ mm}$	16.58 %
$+0.96 \pm 4.76 \text{ mm}$	$2.54 \pm 4.14 \text{ mm}$	40.06 %
$+1.51 \pm 5.89 \text{ mm}$	$3.56 \pm 4.93 \text{ mm}$	21.26 %
$+0.98 \pm 4.59 \text{ mm}$	$2.68 \pm 3.85 \text{ mm}$	19.38 %

TABLE 5.14 – Erreurs mesurées dans les cartes de profondeur de test du jeu de données YCB-VIDÉO [126].

présente les points de vue. Les poses capturées sont très proches les unes des autres le long de ces chemins, et comparé à LINEMOD et T-LESS, une plus petite partie de l'espace des poses est couvert par YCB-Vidéo. Ce type de séquence correspond bien au cadre applicatif des auteurs : la préhension robotique. Mais il correspond mal à la détection 3D à partir d'une seule image, qui ne tirera pas parti de corrélations entre les images d'une même séquence. Et comme certaines poses ne sont pas du tout représentées, on ne peut pas garantir la fiabilité du système de détection 3D à ces poses.

Les auteurs ont bien cadré les objets, et on observe que les objets apparaissent le plus souvent au centre des images (biais central) comme le montre la figure 5.17. Comme pour LINEMOD+OCCLUDED, certains objets apparaissent hors-champs, comme on peut le voir dans les tableaux 5.11 et 5.12.

Les auteurs ont filmé de nombreuses images. Ces images sont annotées semi-automatiquement comme nous l'avons expliqué ci-dessus. En particulier la pose des objets est annotée manuellement seulement dans la première image. Puis elle est propagée dans les images suivantes grâce à une technique de suivi actif basée sur les cartes de profondeur. Et enfin, elle est corrigée entre toutes les images pour minimiser l'erreur globale. Les tableaux 5.13 et 5.14 présentent les erreurs de recalages entre les cartes de profondeur photographiées et celles que nous avons synthétisées avec les étiquettes. L'erreur moyenne globale est de 1.11 mm , ce qui est excellent. Cette faible erreur est due au fait que la méthode d'annotation semi-automatique optimise globalement la même mesure entre les cartes de profondeur photographiées et celles synthétisées avec les poses étiquettes. Les cartes de profondeur peuvent contenir des artefacts ou du bruit. Quand nous mesurons l'erreur de recalage nous ignorons les pixels "aberrants" dont l'erreur est supérieur à 5 cm . Nous calculons aussi la proportion de pixels ignorés de cette façon. En moyenne, ils sont 20.60% , ce qui est plus que tous les jeux de données déjà présentés. Pour certains objets comme le thon en boîte, leurs proportions atteignent les 41.98% . Les cartes de profondeur de YCB-VIDÉO sont plus bruitées que celle de LINEMOD et T-LESS car la caméra est en mouvement. Mais cela n'explique pas les proportions atteintes pour certains objets. La seule autre explication est que le protocole utilisé par les auteurs a donné de mauvais résultats pour certaines images. Cela n'est probablement pas dû à la première étape d'initialisation manuelle, sinon les résultats seraient bien pires. Il reste donc comme possibilité une défaillance de la technique de suivi actif, peut-être de la dérive, ou lors de la phase d'optimisation globale.

Les auteurs de BOP [54, 53] qui utilisent YCV-VIDÉO font aussi état de certaines étiquettes de mauvaise qualité :

"The ground-truth 6D object poses are of a low quality in some images from the original test set. For the BOP Challenge 2019, we have manually selected 75 images (with higher-quality ground-truth poses) from each of the 12 test scenes."

ReadMe inclue dans l'archive de base de la version BOP[54, 53] de
YCV-Vidéo[126].

Traduction :

"Les étiquettes de pose sont de basse qualité dans certaines images de tests. Pour BOP 2019 nous avons manuellement sélectionné 75 images (avec des étiquettes de meilleur qualité) parmi les 12 scènes de tests."

De plus, la vidéo à trente images par seconde produit une forte redondance d'images. À cause de ces deux points seuls, soixante-quinze images par vidéo de test ont été retenues dans BOP. Le même problème est très probablement présent dans les images d'entraînement.

5.5 Autres jeux de données

5.5.1 ITODD (MVTec ITODD)



FIGURE 5.19 – Objets du jeu de données MVTEC ITODD [29].

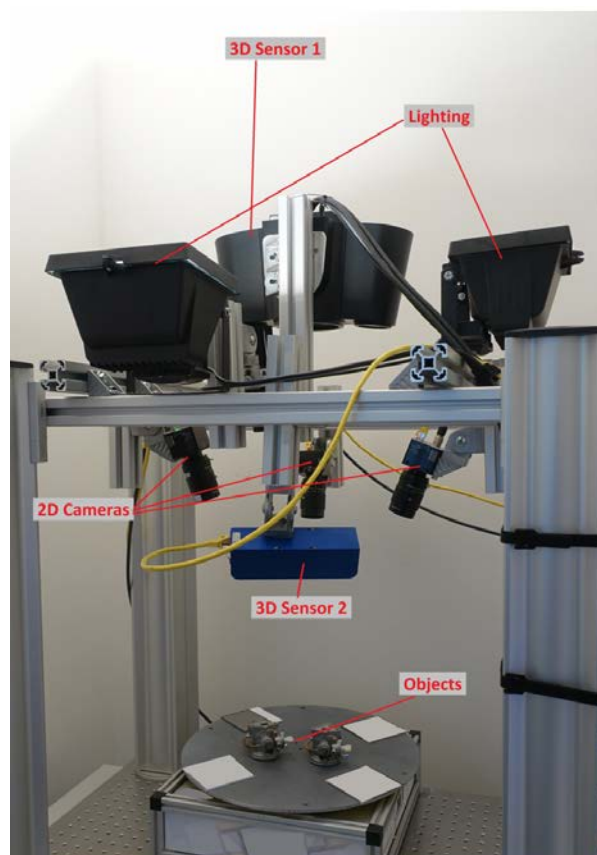


FIGURE 5.20 – Matériels utilisé pour l’acquisition du jeu de données MVTEC ITODD [29].

ITODD [29] est un jeu de données publique produit par l’entreprise MVTec. Il est à destination des industriels qui souhaitent développer et tester des applications de détection 3D. Dans ce jeu de données, cinq caméras sont utilisées pour photographier vingt-huit objets. Deux des caméras utilisées sont des caméra 3D, une haute précision ($100\mu m$) et une de précision moyenne ($1\text{to}2mm$), et les trois autres prennent des images en noir et blanc de haute résolution ($8Mpx$). Les objets présentés en figure 5.19 présentent des variétés de formes : complexes ou simples, plats ou volumineux, détaillés ou non, longs ou compacts, et avec des tailles allant de $2.4cm$ à $27cm$. Ils présentent aussi des facteurs de difficulté comme des surfaces réfléchissantes ou mates et des symétries. Les images sont photographiées par le haut, comme si une caméra était suspendue au-dessus d’un tapis convoyeur. Plusieurs objets d’une même catégorie sont parfois visibles dans une image.

Ce jeu de données s’applique bien à des applications de robotique comme le tri en ligne ou des robots doivent saisir des pièces. Il ne correspond pas bien à notre cadre applicatif : la réalité mixte. Les caméras utilisées sont aussi beaucoup plus précises que celle des casques de réalité mixte ou de tout produit grand public.



FIGURE 5.21 – Objets du jeu de données HOME BREWED-DB [62].

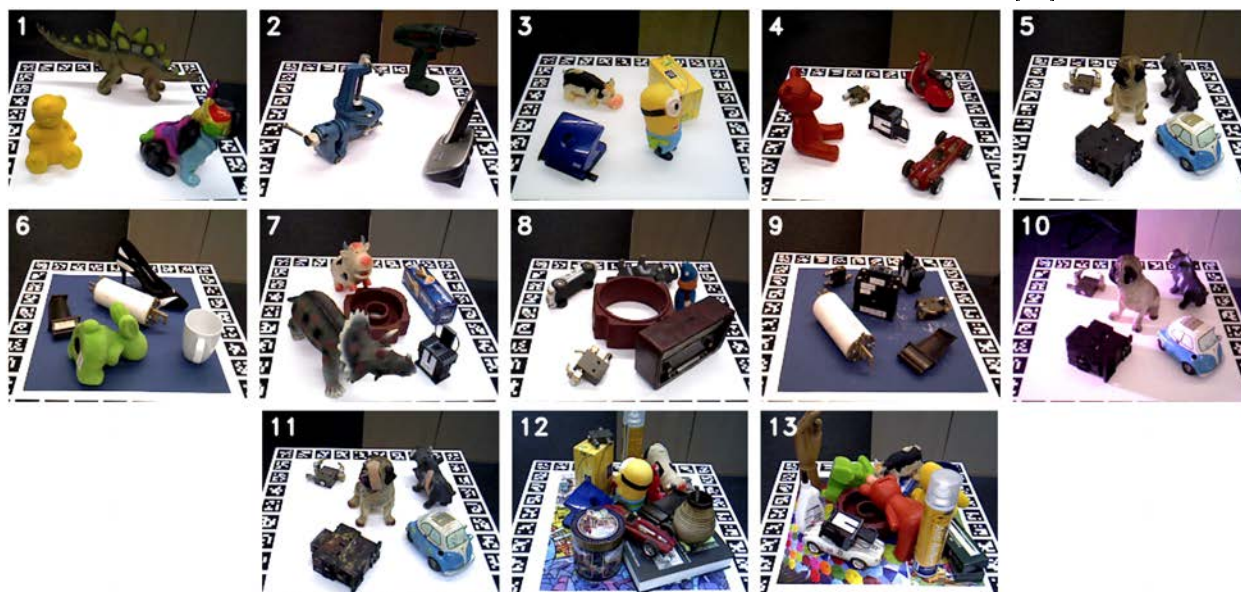


FIGURE 5.22 – Séquences du jeu de données HOME BREWED-DB [62].

5.5.2 Homebrew

Homebrew [62] est un jeu de données produit par Roman Kaskman, Sergey Zakharov, Ivan Shugurov et Slobodan Ilic. Il présente trente-trois objets à différentes vocations comme des jouets, du quotidien ou encore industriels. Ce jeu de données, comme LINEMOD, ne propose aucune image d'entraînement. Seuls les modèles 3D doivent être utilisés pour l'entraînement. Des séquences de tests et de validations sont proposées pour évaluer les méthodes selon plusieurs critères. Les premières scènes présentent trois ou quatre objets avec peu d'occlusions. Les suivantes présentent des objets plus difficiles avec des symétries et des changements de luminosité. Enfin, les dernières séquences présentent un grand nombre d'objets avec beaucoup d'occlusions. Trois séquences en particulier proposent les mêmes objets avec de forts changements de luminosité, aussi bien en terme d'intensité que de direction, et des changements d'apparences (pansements collés sur l'objet, objets repeints). Ces séquences où l'apparence des objets changent sont dites d'adaptation de domaine. Par exemple la séquence 11 de la figure 5.22 montre une variation du chien précédemment vue dans la séquence 5. Un pansement est collé sur la tête du chien.

Bien que plus récent, ce jeu de données est très proche de LINEMOD. Sa contribution majeure est l'ensemble des séquences d'adaptation de domaine qui ne rentre pas dans le cadre de cette thèse.

5.5.3 Falling Things



FIGURE 5.23 – Image du jeu de données FALLING THINGS [116].

Falling Things [116] est un jeu de données 100% synthétique développé par Jonathan Tremblay, Thang To et Stan Birchfield de NVIDIA. Le jeu de données utilise les objets de YCB-VIDÉO [14, 16, 15, 126]. Les objets sont importés dans le moteur 3D "Unreal Engine 4". Ce moteur 3D permet d'obtenir un niveau de détail photoréalistique. Les objets sont disposés dans trois scènes différentes (une cuisine, une forêt et un temple) autour de quinze emplacements sélectionnés manuellement (sur un plan de travail, sur le sol carrelé, sur un canapé, etc.). Dans chaque image synthétisée, les objets sont "lâchés" tête haute aléatoirement autour d'un emplacement donné, d'où le nom "falling things". Des images couleur, cartes de profondeur, masques de segmentation, d'instance et paramètres de pose sont calculés et exportés pour chaque image synthétisée. Les données sont hébergées à l'adresse suivante sur le site de NVIDIA et le logiciel d'extension sur le GitHub NVIDIA. Ce jeu de données constitue une très bonne base d'images synthétiques en combinaison de YCB-VIDÉO.

Dans cette thèse nous nous sommes concentrés sur les données réelles. Nous n'avons donc pas utilisé ce jeu de données.

5.5.4 RU-APC (Rutgers APC)

RUTGERS APC RGB-D DATASET [96] a été développé par Colin Rennie, Rahul Shome, Kostas E. Bekris et Alberto F. De Souza. Son but est de fournir à la communauté un jeu de données permettant d'améliorer la perception robotique et en particulier les applications de préhension robotique. Le jeu de données se place dans un scénario de prise ou de mise en rayonnage d'objets par un robot comme on peut le voir à la figure 5.24. Un robot doté d'une caméra RGB-D et d'un préhenseur est placé devant un rayonnage contenant plusieurs objets. Encore une fois, on retrouve les objets de YCB [14, 16, 15]. Le but est de retrouver la pose des objets pour que le robot puisse effectuer des calculs de cinématique inverse et saisir ces objets. L'application cible est typiquement celle d'un robot qui remplirait les rayons d'un supermarché ou même d'un magasin d'entreprises ou d'usines. Ou encore, un robot qui remplirait des paniers, par exemple de commandes internet, en saisissant les objets en rayon.



(1) Le robot devant le rayonnage.

(2) Le point de vue du robot.

FIGURE 5.24 – Le robot utilisé dans le jeu de données Rutgers APC [96] et une image illustrant son point de vue sur le rayonnage et les objets.

Le jeu de données est disponible sur le site de l'université de Rutgers et au sein du challenge BOP [54, 53].

Ce jeu de données ne correspond pas à notre cadre applicatif. Nous nous sommes concentrés sur YCB-VIDÉO qui propose les mêmes objets et qui est plus utilisé dans l'état de l'art.

5.5.5 IC-BIN



FIGURE 5.25 – Objets du jeu de données IC-BIN [28].

Ce jeu de données a été proposé en 2016 par Andreas Doumanoglou, Rigas Kouskouridas, Sotiris Malassiotis et Tae-Kyun Kim en même temps que leur méthode de détection 3D appelée *"Recovering 6D Object Pose and Predicting Next-Best-View in the Crowd"* [28]. Le jeu de données est disponible sur GitLab. Il y a peu de détails sur le jeu de données dans l'article. Tous ce qui est dit, c'est que deux types de scénarios sont considérés. Dans le premier, des objets du quotidien sont disposés sur une table. Ils sont donc dans une position verticale classique et les seuls occlusions sont dues à la perspective avec la caméra. Et dans le second, les objets sont empilés dans un carton. Il y a donc beaucoup d'occlusions et ils ne ressemblent pas droits.

Après avoir téléchargé le jeu de données, il se trouve que le premier scénario comporte douze scènes, chacune photographiée moins de trente fois (352 images au total). Le second scénario inclut trois scènes de ~ 60 images chacune (183 au total). Au total, le jeu de données

Objet(s)	#images	#instance
Camera	981	2943 ($\times 3$)
Coffe Cup	502	1506 ($\times 3$)
Joystick	839	1678 ($\times 2$)
Juice	557	1671 ($\times 3$)
Milk	289	867 ($\times 3$)
Shampoo	605	1210 $\times 2$
Tous	3773	9875

TABLE 5.15 – Nombre de photographies de chaque objet du jeu de données IC-MI [114]

comporte 535 images. Ce jeu de données a été conçu pour tester une méthode basée sur des templates comme linemod [51] et une forêt de Hough.

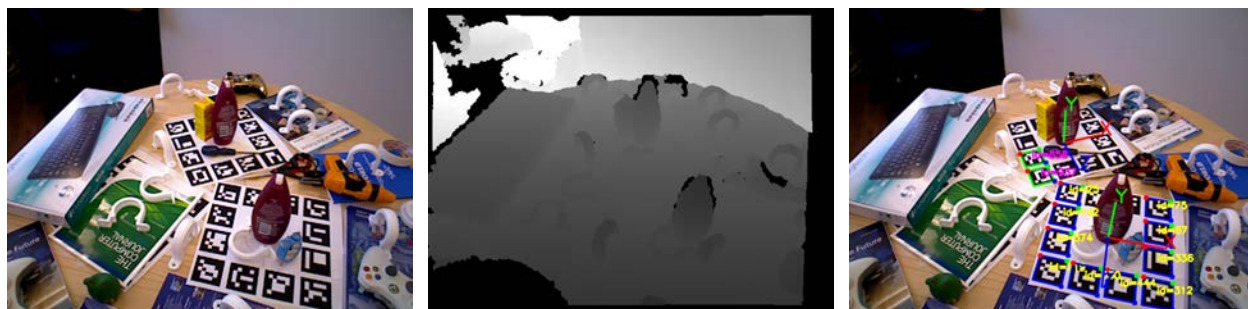
535 images ne sont pas suffisantes pour entraîner des réseaux de neurones. C'est pourquoi ce jeu de données est aujourd'hui utilisé seulement comme ensemble de test. Les réseaux sont entraînés sur des données synthétiques, fournies par exemple par BOP [54, 53]. Ce protocole ne rentre pas dans le cadre de cette thèse. Nous n'avons donc pas utilisé ce jeu de données.

5.5.6 IC-MI



FIGURE 5.26 – Objets du jeu de données IC-MI [28].

IC-MI est une jeu de données produit en 2014 par Alykhan Tejani, Danhang Tang, Rigas Kouskouridas et Tae-Kyun Kim chercheurs au collège impérial de Londres. Il a été développé dans le cadre de leur projet de détection de la pose grâce à des forêts de Hough intitulées "*Latent-Class Hough Forests for Object Detection and Pose Estimation*" [114] publié à ECCV 2014. Le jeu de données est hébergé à l'adresse suivante : <http://rkouskou.gitlab.io/>



(1) Photographie couleur

(2) Carte de profondeur

(3) Annotations ArUco

FIGURE 5.27 – Exemple d'une image couleur et d'une carte de profondeur du jeu de données IC-MI [114]

`research/LCHF.html`. six objets sont présentés à la figure 5.26. Chaque objet fait l'objet d'une mise en scène propre à sa catégorie comme pour LINEMOD. Mais plusieurs instances de ce même objet sont visibles à l'image. Les nombres d'images et d'instances d'objets par image sont présentés dans le tableau 5.15. Les objets sont placés sur des feuilles A4 où sont imprimés des marqueurs ArUco qui permettent de retrouver leurs poses. Un exemple d'image est présenté en figure 5.27.

Par rapport à LINEMOD [50], qui fut introduit en 2013, ce jeu de données n'apporte qu'une seule contribution qui est d'avoir l'objet photographié présent plusieurs fois dans l'image. Mais dans le même temps, ce jeu de données comprend moins d'objets et moins d'images pour chacun d'eux. De même que pour LINEMOD, ce jeu de données a été intégré à BOP [54, 53] comme ensemble de test. L'entraînement se fait alors sur des rendus photo-réalistes, dont la qualité est impactée directement par celle des modèles 3D, qui dans le cas de ce jeu de données n'est pas parfaite comme on peut le voir à la figure 5.26. Les auteurs de BOP ont même du retravailler les modèles dont les maillages n'étaient pas complets. Nous nous sommes concentrés sur LINEMOD dans nos expériences.

Conclusion

On voit que chaque jeu de données a eu pour but de fournir plus d'images d'entraînements et de tests. Pour arriver à ce but, tout en évitant une annotation manuelle de chaque image, deux techniques sont principalement utilisées. La première utilise des marqueurs type ChArUco imprimés sur un plateau pour retrouver la pose des objets placés sur ce dernier. Cette technique est utilisée dans LINEMOD et T-LESS. La deuxième propage la pose annotée manuellement dans quelques images d'une séquence, dans les autres images de la séquence. Cette technique est utilisée dans LINEMOD+OCCLUDED et YCB-Vidéo.

Chaque technique a ses avantages et ses défauts.

Pour utiliser les marqueurs, il suffit de mesurer une fois la position et l'orientation des objets sur le plateau. On peut faciliter cette étape avec une vue temp réel dessinant le modèle 3D par dessus l'objet filmé. La pose est alors calculée automatiquement dans toute les images photographiées par la suite, du moment que la disposition des objets n'est pas modifiée. Cette technique permet d'obtenir une erreur de pose de l'ordre de 5, *mm* pour LINEMOD à 10 *mm* pour T-LESS avec moins de 8% de pixels ignorés. Ce qui reste dans les marges compte tenu de la nature des cartes de profondeur et des caméras utilisées pour les acquisitions. Mais sans l'aide d'un plateau tournant et éventuellement d'un bras robotisé supportant la caméra, il est difficile d'acquérir beaucoup d'images car il faut les photographier une par une. C'est d'ailleurs là une des différences entre LINEMOD et T-LESS : un plateau tournant est utilisé pour photographier T-LESS. Le deuxième désavantage, et de loin le plus problématique, est que le plateau apparaît à l'image. Il doit impérativement être effacé lors de la préparation des images pour ne pas biaiser l'entraînement. Généralement, cela se traduit pas la suppression pure et simple de tout sauf l'avant plan de l'objet. De ce fait, tout le contexte est perdu. De plus, l'objet peut être tronqué si son masque de segmentation est imprécis, ou si un autre objet non-annoté l'occulte. Dans ce dernier cas, l'objet occultant sera inclus dans l'avant-plan par inadvertance.

La technique utilisant le suivi actif peut être utilisée sur des vidéos ou des photos déjà acquises. Elle demande d'annoter précisément un certain nombre d'images. Cette étape est aidée par le même type d'interface que précédemment mais on ne peut s'aider du plateau. Dans une seule, image il est difficile d'annoter précisément la distance entre l'objet et la caméra, puisque celle-ci n'est visible à l'image que par un faible changement d'échelle de l'objet. C'est pourquoi cette étape doit être répétée depuis plusieurs points de vue pour

corriger cette distance. Après quoi, la pose des objets est propagée aux autres images automatiquement. Cette méthode utilise une technique qui minimise l'erreur de pose dans toute les images de la séquence. L'erreur moyenne est donc comparable à celle avec un plateau, voir meilleure. Néanmoins pour certaines images où des objets sont déplacés, ou si un quelconque facteur perturbe l'algorithme, l'erreur est désastreuse. Pour LINEMOD+OCCLUDED l'erreur est comparable en terme de distance moyenne : 5.54 mm comparé à 5.07 mm . Mais il y a plus de valeurs aberrantes : 12.15% contre 7.30% , soit plus de 1.66 fois plus ce qui traduit effectivement de l'échec du suivi dans certaines images. Vu que la séquence de LINEMOD+OCCLUDED ne comporte que 1215 images ils ont pu identifier chaque image où l'algorithme à échoué, corriger manuellement les étiquettes et relancer l'algorithme. Les étiquettes ont d'ailleurs été corrigées d'avantage dans la version BOP de ce jeu de données. Pour YCB-Vidéo qui propose de très nombreuses images l'erreur moyenne encore mieux minimisée est de 2.80 mm . Mais beaucoup plus de pixels sont ignorés dans les calculs : 20% . Cette valeur qui s'élève même à 42% pour la petite pince révèle que certaines images ont des étiquettes complètement fausses car on est bien au delà des marges liées à l'acquisition des cartes de profondeur.

Peut importe la méthode d'acquisition, les jeux de données d'estimation des six degrés de liberté ont tous un point commun : ils ont tous un écart sémantique important entre images d'entraînement et de test. LINEMOD et LINEMOD OCCLUDED ne proposent tout simplement pas d'images d'entraînements. T-LESS propose des images d'entraînements très contrôlées. Les images de tests sont extrêmement difficiles avec des poses inconnues et des occlusions qui se rajoutent aux difficultés propres aux objets. Les séquences de tests de YCB VIDÉO sont inédites et les objets y sont disposés différemment par rapport aux séquences d'entraînements.

Conclusion de la partie

Dans cette partie nous avons étudié les méthodes existantes de classification d'images (1D), de localisation d'objets (2D) et d'estimation des six degrés de liberté (3D). L'état de l'art semble indiquer que les réseaux de neurones convolutionnels (CNN) sont les méthodes qui donnent les meilleurs résultats sur ces trois tâches. Les CNNs sont capables d'associer des probabilités exprimant la présence ou l'absence d'objets dans une image. Ils sont aussi capables de prédire les rectangles encadrants de ces objets et même un masque de segmentation complet qui associe une probabilité à chaque pixel. Et enfin ils sont capables de prédire les six degrés de liberté directement en 3D, ou en 2D grâce à l'utilisation de PnP. Ce domaine est en pleine expansion et de nouvelles architectures, paramétrisation de la pose et expression de fonction de coûts sont régulièrement proposés.

Nous avons aussi présenté dans cette partie les jeux de données pour ces trois tâches. Les jeux de données pour la classification d'images et pour l'apprentissage profond se sont énormément développés dès 2010 grâce à ILSVRC puis COCO. Des énormes collections d'images ont été rassemblées et annotées grâce à l'effort collectif des internautes. Grâce à ces jeux de données, les CNNs pour la classification d'images se sont développés et ont grandement progressé. L'erreur top-5 a été passée de 17.0 % pour AlexNet en 2012 à 0.055 % pour Xception en 2017. Ces jeux de données ont inclus dès 2013 les rectangles encadrants des objets. Par la suite, les résultats des réseaux pour la localisation d'objets se sont considérablement améliorés. La précision moyenne (box AP) est passée de 19.7 % pour Fast R-CNN en 2015, à 37.5 % pour Mask R-CNN en 2018 et à 55.8 % pour YOLO-V4 en 2021. Dans la même période les jeux de données pour l'estimation des six degrés de liberté se sont peu développés. L'annotation manuelle d'images sans marqueurs n'est pas envisageable car elle demanderait trop de temps. De plus les erreurs d'annotation seraient probables. LINEMOD et T-LESS qui utilisent des marqueurs et n'ont pas été conçus pour l'apprentissage profond, continuent donc d'être utilisés. Il a fallu attendre YCB-VIDÉO en 2018 pour voir un jeu de données conçu pour le réseau de neurones. Il a été rendu possible grâce à un nouveau protocole d'annotations semi-automatiques. La précision des résultats (sur LINEMOD) est alors passée de 43.6 % pour BB8 en 2017 à 88.7 % pour PoseCNN (auteurs de YCB VIDÉO) en 2018 puis 95.2 % pour DPOD en 2019.

On voit donc bien que des jeux de données adaptés sont essentiels pour permettre le développement de nouvelles méthodes plus précises. Deux constats sont clairs. Premièrement, la plupart des méthodes d'estimation des six degrés de liberté récentes ont résolu le jeu de données LINEMOD, selon les mesures de performances utilisées aujourd'hui. Ce dernier n'offre, aujourd'hui, pas assez de challenges. Deuxièmement, tous les jeux de données considérés ont des limitations de part leurs protocoles d'acquisitions. Certains n'ont pas d'images d'entraînement ou bien des marqueurs sont visibles dans les images. D'autres ont des problèmes de précision des étiquettes. Un nouveau jeu de données pourrait donc permettre de résoudre ces problèmes tout en offrant plus de possibilités.

Troisième partie

Évaluation sur les jeux de données
existants

Introduction

Dans la partie précédente nous avons listé les méthodes de classification d'images, de localisation d'objets, et d'estimation des six degrés de liberté ainsi que les jeux de données utilisés dans la littérature. Ces méthodes sont le fruit d'un développement actif au sein de tâches en pleines évolutions. Ils nous a donc été difficile de trouver des chiffres permettant de les comparer objectivement, surtout pour l'estimation des six degrés de liberté. Le protocole de préparation des images est souvent très différent. Les mesures utilisées peuvent aussi varier en fonction des objets considérés. Et parfois les résultats sont communiqués sur un jeu de données différent.

Nous avons souhaité entraîner ces méthodes sur les trois jeux de données les plus courants de l'état de l'art : LINEMOD, T-LESS et YCB-VIDÉO avec plusieurs objectifs en tête. Premièrement, nous souhaitons calculer les mêmes mesures pour avoir les mêmes chiffres sur toutes les méthodes. Deuxièmement, les méthodes sont toutes testées d'abord avec le même protocole de préparation des images. Ce protocole de base permet de comparer les résultats obtenus avec les mêmes images d'entraînement et de test. Si la méthode le permet, la préparation des images d'entraînement est ensuite étendue avec plus d'augmentations. Troisièmement, nous détaillons la mise en œuvre technique de ces méthodes et l'entraînement des réseaux. Enfin, ces entraînements ont parfois suscité des questions et nous avons approfondi nos entraînements et nos résultats pour y répondre.

Cette partie de la thèse retrace ces expériences. Le premier chapitre est consacré aux données d'apprentissage : images et étiquettes pour chaque tâche. Il détaille en particulier tous les calculs des étiquettes. Le second chapitre est consacré aux mesures et aux fonctions de coûts associées à chaque étiquette. Tous les calculs sont expliqués et des exemples visuels sont présentés. Le troisième chapitre est consacré aux techniques de préparations et d'augmentations des images avec des exemples visuels. Le dernier chapitre présente les résultats obtenus et les comparaisons établies.

Chapitre 6

Données d'apprentissages

6.1 Introduction

Ce premier chapitre est consacré au données d'apprentissage. Nous apportons des illustrations et formalisons la notation qui sera utilisé dans tous le reste de la thèse. Pour commencer présentons rapidement les images couleur et les cartes de profondeur dans un chapitre. Ce sont les seules données utilisées par les réseaux de neurones que nous considérons pour établir leurs prédictions. Pour nos entraînements nous n'utilisons que les images couleur. Les cartes de profondeur sont présentées car elles sont utilisées dans certain calculs d'étiquettes ou mesures d'erreurs. Les trois chapitres suivants détaillent les étiquettes pour les trois tâches que nous avons considérées. Celui consacré à la classification d'images présente notamment les vecteurs logiques. Pour la localisation d'objets nous verrons comment calculer les rectangles encadrants et comment les utiliser pour construire d'autres étiquettes. Et enfin pour l'estimation des six degrés de liberté nous détaillerons les étiquettes calculés à partir de la matrice de rotation 3D et du vecteur de translation 3D.

6.2 Images

6.2.1 Caméras



(1) Microsoft Kinect ®



(2) Intel RealSense d435i ®

FIGURE 6.1 – Exemples de caméras de profondeurs.

Les images sont la seule donnée à disposition de notre module de détection pour inférer la catégorie des objets présents à l'image, leurs positions dans l'image et leurs orientations. Les

jeux de données, pour l'estimation des six degrés de liberté, sont généralement photographiés grâce à des caméras 3D qui en plus des images couleur, capturent des cartes de profondeur. Deux exemples de caméras de profondeurs sont présentés à la figure 6.1.

6.2.2 Images Couleurs

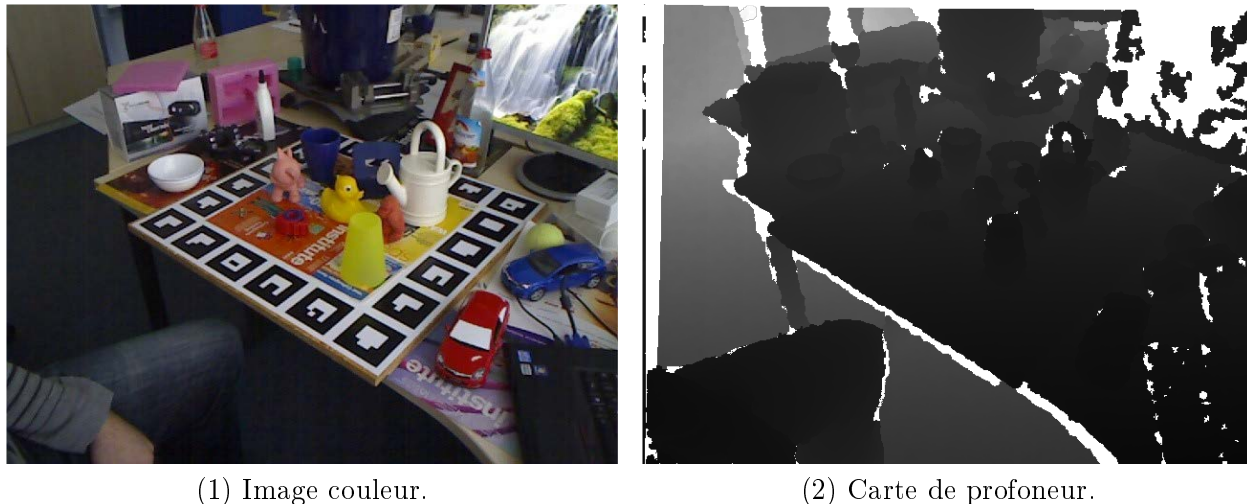


FIGURE 6.2 – Exemple d'une image couleur et d'une carte de profondeur du jeu de données LINE-MOD photographiée par une Microsoft Kinect [®].

Les images sont des grilles de pixels rectangulaires où chaque pixel comporte trois valeurs : le rouge, le vert, le bleu. On peut aussi ajouter une quatrième valeur pour modéliser la transparence : l'alpha. Cette dernière est utile pour travailler les images et effacer l'arrière-plan mais n'est pas utilisée par les réseaux de neurones. Toutes les valeurs sont généralement échantillonnées sur 8 bits. Elles sont donc comprises entre 0 et 255, soit 256 nuances pour un canal et 16 777 216 pour trois canaux. Lorsque les images sont préparées pour le réseau de neurones, on convertit ces valeurs entières en nombres flottants compris entre 0.0 et 1.0 encodés sur 32 bits.

Les jeux de données comprennent plusieurs images. Pour définir une image dans cette thèse, nous utilisons la notation suivante :

$$I_i = \{ (rouge, vert, bleu, alpha)_{u,v} \forall u \in [0, W[\wedge v \in [0, H[\} \quad (6.1)$$

Où $i \in [0, N]$ est l'indice de l'image dans le jeu de données.

6.2.3 Cartes de profondeurs

Les pixels des cartes de profondeur ne prennent qu'une seule valeur. Elle correspond à la distance à l'objet. Si la distance est mesurée par rapport au plan image, on parle de carte de profondeur. Et si elle est mesurée par rapport à l'origine de la caméra, on parlera de carte de distances. La distance est stockée comme un nombre flottant sur 16 ou 32 bits. Pour pouvoir les traiter comme de simples images et faciliter leur stockage, on peut aussi les convertir en nombres entiers sur 8, 16 ou 32 bits. Pour passer de la représentation entière à la représentation flottante, on multiplie par une valeur d'échelle de profondeur ("*depth scale*" en Anglais) que nous noterons $\lambda_{\mathbb{P}}$, et qui est propre à la caméra utilisée.

$$P_i = \{ (distance)_{u,v} \forall u \in [0, W[\wedge v \in [0, H[\} \quad (6.2)$$

Les caméras 3D sont équipées de plusieurs objectifs : un pour les images couleur et deux pour les cartes de profondeur. Vu que ces objectifs sont espacés de quelques dizaines de millimètres, les images obtenues ne sont pas alignées. Le pixel de coordonnées (u, v) dans l'image couleur ne correspond pas au pixel (u, v) dans la carte de profondeur. Les deux images peuvent même ne pas avoir les mêmes résolutions. Heureusement, ces caméras sont fournies avec des outils logiciels qui permettent d'aligner les deux images. Les jeux de données que nous allons considérer sont fournis avec des images pré-alignées et de mêmes résolutions.

6.3 Étiquettes pour la détection 1D

Nom	Index	Vecteur Logique
Ape	0	[●○○○○○○○○○○○○○○○○]
Benchvise Blue	1	[○●○○○○○○○○○○○○○○]
Bowl	2	[○○●○○○○○○○○○○○○]
Cam	3	[○○○●○○○○○○○○○○○○]
Can	4	[○○○○●○○○○○○○○○○○○]
Cat	5	[○○○○○●○○○○○○○○○○]
Cup	6	[○○○○○○●○○○○○○○○○○]
Driller	7	[○○○○○○○●○○○○○○○○]
Duck	8	[○○○○○○○○●○○○○○○○○]
Eggbox	9	[○○○○○○○○○●○○○○○○]
Glue	10	[○○○○○○○○○○●○○○○○]
Holepuncher	11	[○○○○○○○○○○○○●○○○]
Iron	12	[○○○○○○○○○○○○○○●○○]
Lamp	13	[○○○○○○○○○○○○○○○●○]
Phone	14	[○○○○○○○○○○○○○○○○●]

FIGURE 6.3 – Exemples des différentes étiquettes pour la détection 1D des 11 catégories d'objets du jeu de données LINEMOD. L'index prends des valeurs comprises entre 0 inclus et 11 exclu. Le vecteur logique est composé de 11 bit où le bit actif est celui dont la position dans le vecteur est égal à l'index de la catégorie.

Premièrement, il est important de préciser la différence entre catégorie d'objet et instance d'objet. Une catégorie d'objet dénote un modèle 3D ou son nom. Lorsque cet objet apparaît à l'image on parle alors d'une instance. S'il apparaît plusieurs fois à l'image, chaque instance est de la même catégorie, mais est identifiée individuellement :

$$\begin{cases} c_{i,j} = c_{i,k} \\ j \neq k \end{cases} \quad (6.3)$$

Détecter plusieurs instances demande de les localiser sur l'image pour pouvoir les différencier. Nous en parlerons dans la prochaine section consacrée aux étiquettes pour la détection 2D. Cette section est consacrée aux étiquettes de détection 1D. Il s'agit donc seulement de reconnaître quels objets sont présents dans une image, indifféremment du nombre d'instances d'objet d'une même catégorie.

6.3.1 Noms des objets

Comme expliqué dans la Partie 4.1 de cette thèse, la détection 1D a pour but de trouver quels objets sont présents dans l'image. Pas leurs modèles 3D, mais juste la liste des objets.

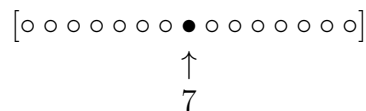


FIGURE 6.4 – Illustration de la construction d'un vecteur logique. La cellule dont la position dans le vecteur correspond à l'index de la catégorie d'objet est "Vrai" alors que les autres sont "Faux". Ici, c'est la septième catégorie qui est présente, la perceuse ("*Driller*" en Anglais).

L'étiquette de base pour la détection 1D est donc la liste des noms des objets. Si les modèles 3D sont disponibles, alors on peut récupérer le modèle 3D d'un objet par association avec son nom.

Les jeux de données pour la détection 1D sont généralement fournis sous la forme de dossiers contenant des images. Chaque dossier donne le nom d'une catégorie d'objet et contient les images où figure cet objet. LINEMOD est fourni selon ce même modèle et les étiquettes de ce jeu de données sont déduites de l'arborescence de fichiers. Pour ces jeux de données, la liste des noms des dossiers correspond donc à la liste des noms des objets. Les étiquettes de LINEMOD sont présentées à la figure 6.3. Si plusieurs objets sont présents à l'image, ils sont généralement listés dans un fichier texte fourni avec l'image.

Les réseaux de neurones ne prédisent pas directement de texte. Il faut donc trouver une autre représentation qui permette d'associer un texte avec un nombre.

6.3.2 Index des objets

On peut utiliser un nombre entier pour représenter la catégorie appelé "index". Ce nombre sera compris entre 0 inclu et le nombre de catégories d'objets exclu $|M|$. La position dans la liste des objets donne l'indice et inversement. Dans cette thèse nous notons l'index du j -ème objet de la i -ème image par $c_{i,j}$:

$$c_{i,j} \in [0, |M|] \quad (6.4)$$

6.3.3 Vecteur Logique

Les réseaux de neurones ne comprennent néanmoins pas non plus la notion d'index. En effet, la sortie d'un réseau de neurones est composée de plusieurs neurones. Chaque neurone exprime une valeur réelle encodée par un nombre flottant à 32 bits. Bien que ce nombre puisse prendre les valeurs de notre index, cela ne serait pas une bonne représentation de la catégorie, car le réseau devrait apprendre à modéliser plusieurs nombres entiers à partir d'un seul nombre flottant. Il est préférable que la dernière couche du réseau de neurones, que l'on appelle "couche de classification", soit composée d'autant de neurones que de catégories d'objets. Chaque neurone est alors chargé de détecter une catégorie d'objet dans l'image. Sa valeur sera la plus grande possible si l'objet dont ils ont la charge est présent, et inversement, elle sera la plus faible possible si ce n'est pas le cas. Ces valeurs sont ensuite normalisées grâce à la fonction *softmax* dont l'équation est rappelée ci-dessous. Cette fonction permet d'obtenir une distribution de probabilités avec des valeurs comprises entre 0 et 1 inclus. La sortie du réseau de neurones pour l'image I_i est alors un ensemble de probabilités, autant que de catégories d'objets, que nous noterons \hat{p}_i :

$$\hat{p}_i = \{ softmax(\hat{p}_{i,k}) \forall k \in |M| \} \quad (6.5)$$

$$softmax(\hat{p}_{i,k}) = \frac{e^{\hat{p}_{i,k}}}{\sum_{l=0}^{|M|} e^{\hat{p}_{i,l}}} \forall l \in [0, |M|] \quad (6.6)$$

Pour entraîner le réseau de neurones, on représente la catégorie de l'objet par un vecteur logique ("*one-hot vector*" en Anglais). Cette représentation contient autant de valeurs que de catégories d'objet (et de sortie du réseau). Si un objet est présent dans l'image, alors la valeur associée à l'objet sera *vrai*, sinon elle sera *faux*. Sa construction est illustrée à la figure 6.4.

Formellement le vecteur logique p_i de l'image I_i aura $|M|$ valeurs :

$$p_i = \{p_{i,k} \forall k \in [0, |M|[\} \quad (6.7)$$

Les cellules de ce vecteur ont pour valeur :

$$p_{i,k} = \begin{cases} vrai & \text{si } k = c_{i,j} \\ faux & \text{sinon} \end{cases} \quad (6.8)$$

Inversement lors de l'inférence avec le réseau de neurones, il est possible de retrouver l'index à partir de probabilités produites par le réseau. Si un seul objet est recherché, on peut utiliser directement l'indice de la plus forte probabilité.

$$\widehat{c}_{i,j} = \operatorname{argmax}_{k \in [0, |M|[\} \{ \widehat{p}_{i,k} \} \quad (6.9)$$

Si plusieurs objets sont présents, on utilise plutôt un seuil de décision. Un objet d'indice k sera considéré comme présent si $p_{i,k} \geq \tau$. Où τ est un seuil de décision fixé manuellement dont on peut optimiser la valeur grâce aux résultats des mesures de performance (présentées plus tard).

6.4 Étiquettes pour la détection 2D

La détection 2D cherche à savoir où se trouvent les objets dans l'image. Ce problème peut être formulé de deux façons différentes. Soit on cherche à savoir dans quelle région de l'image se trouve l'objet. Par exemple, on estime des rectangles encadrants. Soit on cherche à savoir à quels objets appartiennent chaque pixel de l'image. Ces deux formulations sont très différentes et n'ont pas du tout les mêmes solutions comme nous l'avons vu dans l'état de l'art.

La première formulation cherche juste à identifier des zones dans l'image qui correspondent à des objets. Cette tâche est appelée détection d'objets. On peut utiliser n'importe quelle forme pour définir la zone à identifier. L'idée est de pouvoir formuler cette région avec le moins de paramètres possibles et, qu'en même temps, la forme de la région soit utilisable, par exemple pour recadrer l'image plus tard. Ces paramètres constitueront alors les étiquettes et on cherchera à les estimer grâce à un réseau de neurones. On peut chercher par exemple un cercle centré sur l'objet, qui sera modélisé par son centre et son rayon, soit trois paramètres. Plus souvent, on cherche le plus petit rectangle encadrant l'objet ("*bounding-rectangle*" en Anglais). Ils est lui modélisé par les coordonnées de son coin supérieur gauche et de son coin inférieur droit, soit quatre paramètres.

La deuxième cherche à segmenter l'image. Les étiquettes utilisées sont similaires à celles pour la détection 1D. Mais au lieu d'attribuer une étiquette à toute l'image, on attribue une étiquette à chaque pixel de l'image. Comme pour la classification d'images, le problème des instances multiples se pose avec ces étiquettes. Pour résoudre cette tâche, les réseaux de neurones encodeurs-décodeurs sont les plus appropriés.

Dans cette section, nous allons voir comment calculer ces étiquettes. Premièrement, nous verrons deux méthodes pour calculer le rectangle encadrant d'un objet dans l'image. Deuxièmement, nous verrons ce qu'est le masque de segmentation et comment le calculer.

$M_{i,j,\mathbb{I}}$		
k	u	v
0	0.4100	0.5478
1	0.5154	0.5143
2	0.4011	0.4845
3	0.4231	0.5305
4	0.4151	0.5362
5	0.4146	0.4690
6	0.3992	0.4993
7	0.4375	0.5369
8	0.5578	0.4910
9	0.4253	0.5503
10	0.3993	0.4819
...		
7911	0.4859	0.6025
<i>min</i>	0.3948	0.3872
<i>max</i>	0.6181	0.6155

(1) Calcul façon tableau.



(2) Représentation graphique.

FIGURE 6.5 – Exemple du calcul du rectangle encadrant à l'aide des points du modèle 3D du Canard de LINEMOD. Le modèle 3D est projeté dans le plan image avant d'effectuer ce calcul. La première colonne du tableau (a) indique les indices des points, noté k . Les deux colonnes suivantes donnent les coordonnées u et v des points dans le repère de l'image. Les minimums et maximums des points en u et en v donnent les coordonnées des coins du rectangle encadrant. L'image (b) illustre ce calcul où les points sont représentés en vert.

6.4.1 Rectangle encadrant

Les rectangles encadrants permettent d'obtenir une région rectangulaire. Elles sont suffisantes pour beaucoup d'applications de réalité augmentée et de compréhension de scènes. Les rectangles peuvent aussi être recadrés et passés par la suite à un autre réseau pour la détection 3D ou tout autre tâche. Nous noterons $br_{i,j,\mathbb{I}}$ le rectangle encadrant l'objet $M_{i,j}$ dans l'image I_i . Il sera formulé comme un tableau de deux points : le coin supérieur gauche ($br_{i,j,0,u,\mathbb{I}}, br_{i,j,0,v,\mathbb{I}}$) et le coin inférieur droit ($br_{i,j,1,u,\mathbb{I}}, br_{i,j,1,v,\mathbb{I}}$) :

$$br_{i,j,\mathbb{I}} = \begin{bmatrix} br_{i,j,0,u,\mathbb{I}} & br_{i,j,0,v,\mathbb{I}} \\ br_{i,j,1,u,\mathbb{I}} & br_{i,j,1,v,\mathbb{I}} \end{bmatrix} \quad (6.10)$$

Dans la plupart des jeux de données pour la détection 3D les rectangles encadrants sont pré-calculés et fournis. Mais ce n'est pas nécessairement le cas. Ci-dessous, nous détaillons deux méthodes pour les calculer, soit grâce à la projection du nuage de points dans l'image, soit grâce au rendu des objets ou à leurs masques de segmentation.

Calcul à partir du nuage de points

La première façon de calculer les coordonnées du rectangle encadrant est d'utiliser les points du modèle 3D. Les coordonnées des coins sont obtenues en projetant le nuage de points de l'objet dans l'image et en trouvant les minimums et maximums en X et en Y. On rappelle que la projection du nuage de points est obtenue grâce à la formule du système optique (voir 2.2.7) :

$$M_{i,j,\mathbb{I}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times M_{j,\mathbb{M}} \quad (6.11)$$

Le résultat est une liste de points en deux dimensions de coordonnées $(M_{i,j,k,u,\mathbb{I}}, M_{i,j,k,v,\mathbb{I}})$. Pour trouver les dimensions du rectangle qui les encadrent, on utilise les quatre formules suivantes qui permettent de trouver les quatre coordonnées des deux coins du rectangle :

$$\left\{ \begin{array}{l} br_{i,j,0,u,\mathbb{I}} = \underset{k \in [0, |M_{i,j,\mathbb{I}}|[}{\operatorname{argmin}} M_{i,j,k,u,\mathbb{I}} \\ br_{i,j,0,v,\mathbb{I}} = \underset{k \in [0, |M_{i,j,\mathbb{I}}|[}{\operatorname{argmin}} M_{i,j,k,v,\mathbb{I}} \\ br_{i,j,1,u,\mathbb{I}} = \underset{k \in [0, |M_{i,j,\mathbb{I}}|[}{\operatorname{argmax}} M_{i,j,k,u,\mathbb{I}} \\ br_{i,j,1,v,\mathbb{I}} = \underset{k \in [0, |M_{i,j,\mathbb{I}}|[}{\operatorname{argmax}} M_{i,j,k,v,\mathbb{I}} \end{array} \right. \quad (6.12)$$

La figure 6.5 donne un exemple visuel du calcul du rectangle encadrant à partir des points du Canard de LINEMOD. On peut y voir en vert une partie des points du nuage qui sont projetés et dessinés sur l'image. Les formules ci-dessus donnent les abscisses et les ordonnées des deux points rouges qui sont les coins du rectangle encadrant.

Calcul à partir du masque de l'objet

Sans le nuage de points, il est possible d'utiliser le masque de segmentation ou le rendu de l'objet. Cette façon de calculer le rectangle encadrant utilise les coordonnées pixels du rendu ou du masque. Elle est donc précise au pixel près seulement.

Le rendu $\widehat{I}_{i,j}$ de l'objet $M_{i,j}$ dans l'image I_i est une image à quatre canaux : rouge, vert, bleu et alpha. Le dernier canal, alpha, est très utile, car s'il est nul pour un pixel, il indique que ce pixel est complètement transparent. Donc, que l'objet n'a pas contribué à la couleur de ce pixel car aucun des rayons de lumière en provenance de l'objet n'ont terminé leur course dans ce pixel. Le masque binaire de segmentation $B_{i,j}$ donne la même information. Un pixel y est *faux* s'il n'appartient pas à l'objet et *vrai* s'il appartient à l'objet. On peut d'ailleurs calculer le masque à partir du canal alpha du rendu :

$$B_{i,j,u,v} = \begin{cases} vrai & \text{si } \widehat{I}_{i,j,u,v,\alpha} > 0 \\ faux & \text{sinon} \end{cases} \quad \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (6.13)$$

Dans le masque on ne considère qu'un seul objet. Il n'y a donc qu'un seul "blob" de pixels dont les valeurs sont "vrai" (égal à 1). Un "blob" est un amas de pixels qui sont connexes. Trouver le rectangle encadrant revient à trouver le rectangle qui encadre ce "blob". Pour ce faire, on considère le masque comme une image à valeurs booléennes. On calcule pour chaque ligne de pixel si elle contient au moins un pixel "vrai", i.e. si l'objet est visible dans cette ligne. Puis on fait la même chose pour les colonnes. On obtient une rangée horizontale de pixels qui décrit si l'objet est présent dans les colonnes, et une rangée verticale qui décrit si l'objet est présent dans les lignes correspondantes. Les indices des premiers et derniers pixels *vrais* dans la rangée horizontale donne les abscisses des deux coins du rectangle. Ceux dans la ligne verticale donnent leurs ordonnées. Ce calcul est illustré à la figure 6.6.

Comparaison

Nous avons vu comment calculer le rectangle encadrant d'un objet, soit à partir de son nuage de points, soit à partir de son masque de segmentation.

La première technique nécessite l'accès au modèle 3D, ainsi qu'aux paramètres intrinsèques et extrinsèques. On peut donc l'utiliser dans les jeux de données pour la détection

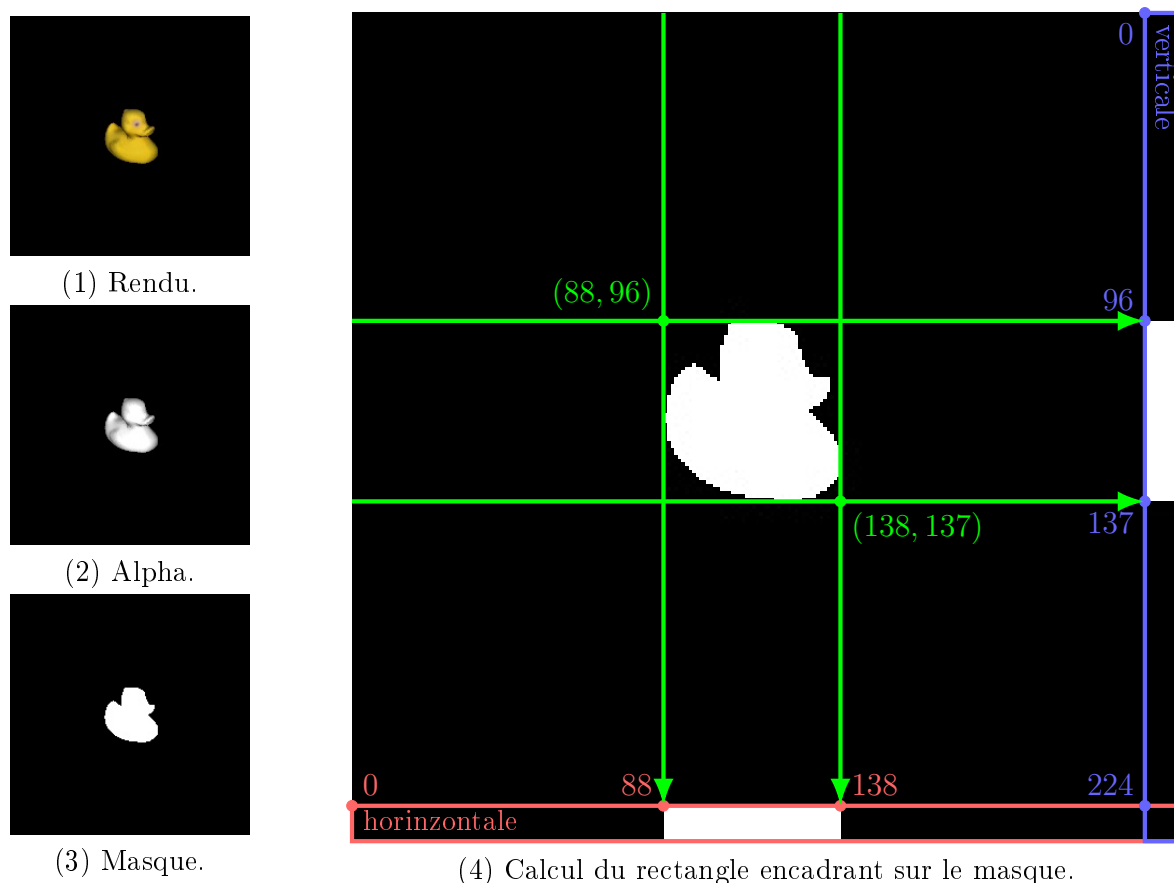


FIGURE 6.6 – Calcul du rectangle encadrant du "canard" à partir du rendu (a). On extrait d'abord le canal alpha du rendu (b). Puis on le binarise pour trouver les pixels non-nul (c). Puis on calcule si l'objet est présent ligne par ligne, ce qui donne le rectangle bleu à droite de l'image (d). On fait la même opération pour les colonnes, ce qui donne le rectangle rouge en bas de l'image (d). Enfin on cherche les indices des premiers et derniers pixels non-nul dans ces deux rectangles. Ces indices, illustrés avec les flèches vertes dans l'image (d), sont les coordonnées des coins du rectangle encadrant. Comme on peut le voir, les flèches vertes encadrent bien le "canard".

3D. Le rectangle encadrant obtenu a une précision flottante inférieure au pixel. Mais son résultat est directement impacté par la qualité du modèle 3D et des paramètres de la caméra.

La deuxième technique peut, elle, être appliquée dès que les masques de segmentation sont disponibles. Ils peuvent eux-même être calculés grâce aux rendus. Le rectangle encadrant obtenu à une précision entière au pixel près. Si les rendus sont utilisés pour calculer les masques de segmentation, alors cette technique est aussi soumise à la qualité des modèles 3D et des paramètres de la caméra.

Maintenant que nous savons comment calculer les rectangles encadrants, voici comment ils sont utilisés par plusieurs méthodes.

Normalisation

Les deux techniques que nous avons vues ci-dessus donnent des rectangles encadrants dont les coordonnées sont formulées en pixels sur une image. Les dimensions d'une image peuvent énormément varier par exemple entre un cadre de 32×32 pixels pour MNIST ou une image en $4K$ de 4096×2160 pixels et bientôt en $8K$. De ce fait, la plage de valeurs qu'un rectangle encadrant peut prendre, varie tout autant. Ceci peut être problématique pour les réseaux de neurones comme nous le verrons plus tard. Pour palier à ce problème,

il est possible de normaliser les coordonnées des rectangles encadrant par rapport à la taille de l'image :

$$\begin{cases} \overline{br_{i,j,0,u,\mathbb{I}}} = br_{i,j,0,u,\mathbb{I}}/W \\ \overline{br_{i,j,0,v,\mathbb{I}}} = br_{i,j,0,v,\mathbb{I}}/H \\ \overline{br_{i,j,1,u,\mathbb{I}}} = br_{i,j,1,u,\mathbb{I}}/W \\ \overline{br_{i,j,1,v,\mathbb{I}}} = br_{i,j,1,v,\mathbb{I}}/H \end{cases} \quad (6.14)$$

Cette normalisation contraint les coordonnées des rectangles entre 0 et 1 ce qui permet d'appliquer une activation *Sigmoid* pour aussi contraindre les prédictions d'un réseau de neurones à sa sortie. Une fonction de coût dont les valeurs sont bornées sera aussi plus facile à intégrer et interpréter.

Rectangles de R-CNN

R-CNN et Fast R-CNN utilisent l'algorithme de recherche sélective pour identifier des régions candidates puis les classifier. Les rectangles ont des coordonnées exprimées en pixels comme expliqué ci-dessus. Pour utiliser ces méthodes, il suffit d'entraîner ou d'affiner l'entraînement de la méthode de classification d'image.

Faster R-CNN remplace la recherche sélective par un sous-réseau de prédiction de région (R.P.N.). Pour la première fois, les rectangles sont prédits grâce à un réseau de neurones qu'il faut entraîner. Ce réseau produit K rectangles (soit $K \times 4$ coordonnées) et $K \times 2$ probabilités indiquant si le rectangle est un objet ou de l'arrière-plan (voir figure 4.93 et section 4.2.4). Les coordonnées des rectangles sont exprimées relativement à des a priori. Les auteurs de Faster R-CNN ont choisi d'utiliser neuf a priori : trois de tailles et, trois de ratios d'aspects (soit neuf combinaisons). Nous dénotons ces a priori (ou ancres) comme :

$$\{\ddagger_{k,\mathbb{I}} \forall k \in [0, 9[\} \quad (6.15)$$

$$\Leftrightarrow \{(\ddagger_{k,cu,\mathbb{I}}, \ddagger_{k,cv,\mathbb{I}}, \ddagger_{k,w,\mathbb{I}}, \ddagger_{k,h,\mathbb{I}}) \forall k \in [0, 9[\} \quad (6.16)$$

Les ancres sont choisies manuellement par les auteurs. Elles sont centrées sur la région considérée par le RPN. Les valeurs de $\ddagger_{k,cu}$ et de $\ddagger_{k,cv}$ dépendent donc de la position de la fenêtre glissante du RPN dans son tenseur d'entrée. Elles sont donc calculées lors de la passe d'inférence. Seul $\ddagger_{k,w}$ et $\ddagger_{k,h}$ constituent des a priori (taille et échelle).

Le RPN prédit des décalages et des déformations des ancres plutôt que de prédire directement des rectangles. Les étiquettes de Faster R-CNN sont donc formulées par rapport aux ancres. Pour chaque rectangle étiquette, on peut donc calculer $k \in [0, 9[$ étiquettes pour Faster R-CNN. Si $br_{i,j,cu}, br_{i,j,cv}$ dénote les coordonnées du centre d'un rectangle étiquette de l'objet M_j dans l'image I_i , que $br_{i,j,k,w}$ dénote sa largeur et $br_{i,j,k,h}$ sa hauteur alors les étiquettes de Faster R-CNN sont calculées comme :

$$\begin{cases} \delta_{i,j,k,cu,\mathbb{I}} = (br_{i,j,k,cu,\mathbb{I}} - \ddagger_{k,cu,\mathbb{I}}) / \ddagger_{k,w,\mathbb{I}} \\ \delta_{i,j,k,cv,\mathbb{I}} = (br_{i,j,k,cv,\mathbb{I}} - \ddagger_{k,cv,\mathbb{I}}) / \ddagger_{k,h,\mathbb{I}} \\ \lambda_{i,j,k,w,\mathbb{I}} = \log(br_{i,j,k,w,\mathbb{I}} / \ddagger_{k,w,\mathbb{I}}) \\ \lambda_{i,j,k,h,\mathbb{I}} = \log(br_{i,j,k,h,\mathbb{I}} / \ddagger_{k,h,\mathbb{I}}) \end{cases} \quad (6.17)$$

Inversement, lors de l'inférence, le RPN prédit un décalage du centre ($\widehat{br_{i,k,cu}}, \widehat{br_{i,k,cv}}$) et des déformations des ancres $\widehat{br_{i,k,w}}$ et $\widehat{br_{i,k,h}}$ sur l'axe U et V respectivement. Les coordonnées

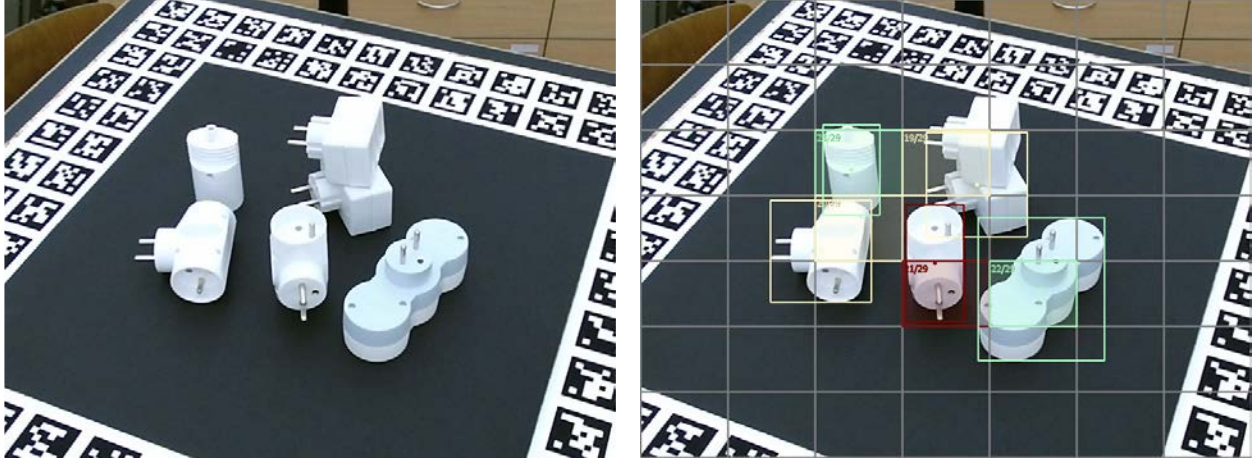


FIGURE 6.7 – Illustration des étiquettes de YOLO V1 pour une image du jeu de données T-LESS. La grille en gris délimite les cellules modélisées par YOLO. Les rectangles sans remplissages représentent les rectangles encadrants des objets inclus dans les étiquettes. Les rectangles avec remplissages identifient les cellules en charge de prédire un objet. Et les points indiquent l'emplacement des centres de rectangles encadrants des objets. Tout les objets ne sont pas inclus dans les étiquettes du fait que certains tombent dans la même cellule.

du rectangle prédit sont ensuite calculées comme :

$$\begin{cases} \widehat{br_{i,k,cu,\mathbb{I}}} = \delta_{i,k,cu,\mathbb{I}} \times \dot{\downarrow}_{k,w,\mathbb{I}} + \dot{\downarrow}_{k,cu,\mathbb{I}} \\ \widehat{br_{i,k,cv,\mathbb{I}}} = \delta_{i,k,cv,\mathbb{I}} \times \dot{\downarrow}_{k,h,\mathbb{I}} + \dot{\downarrow}_{k,cv,\mathbb{I}} \\ \widehat{br_{i,k,w,\mathbb{I}}} = \exp(\lambda_{i,k,w,\mathbb{I}}) \times \dot{\downarrow}_{k,w,\mathbb{I}} \\ \widehat{br_{i,k,h,\mathbb{I}}} = \exp(\lambda_{i,k,h,\mathbb{I}}) \times \dot{\downarrow}_{k,h,\mathbb{I}} \end{cases} \quad (6.18)$$

L'architecture utilisée par Faster R-CNN pour extraire des descripteurs est VGG. Cette dernière produit un tenseur de dimensions 7×7 après la dernière couche de convolution. Il y a donc 5×5 emplacements possibles pour le RPN qui a un champ réceptif de 3×3 . Au total, on obtient donc $5 \times 5 \times 9 = 225$ propositions de régions en sortie du RPN. Les détections finales sont obtenues par application de la suppression des non-maximaux.

Rectangles de YOLO

La première version de YOLO n'utilise pas d'ancre, mais elle découpe l'image selon une grille. Chaque cellule de la grille est responsable de prédire le rectangle d'un objet. La position du rectangle prédit est relative au centre de la cellule qui prédit ce rectangle. Et la taille du rectangle est prédite proportionnellement à la taille de l'image. Donc, en chaque cellule, le réseau prédit donc un décalage $(\widehat{\delta_{i,k,cu,\mathbb{I}}}, \widehat{\delta_{i,k,cv,\mathbb{I}}})$ par rapport au coin supérieur gauche $(\widehat{I_{i,k,u_0,\mathbb{I}}}, \widehat{I_{i,k,v_0,\mathbb{I}}})$ de la cellule $I_{i,k}$ et des déformations des dimensions de l'image I_i notées $(\lambda_{i,k,w,\mathbb{I}}, \lambda_{i,k,h,\mathbb{I}})$.

Les auteurs considèrent qu'une cellule est responsable de prédire le rectangle encadrant d'un objet, si le centre du rectangle encadrant est dans la cellule. Ils ne précisent néanmoins pas comment traiter le cas où plusieurs objets tombent dans la même cellule. Nous avons choisi de privilégier l'objet qui occupe la plus grande aire à l'image. On peut aussi choisir l'objet d'avant plan ou l'objet le plus visible.

Les étiquettes de YOLO sont présentées à la figure 6.7. Pour la cellule $I_{i,k}$ de l'image I_i et une fois un objet $M_{i,j}$ et son rectangle encadrant $bb_{i,j}$ choisis, les étiquettes pour entraîner

YOLO sont calculées comme :

$$\begin{cases} \delta_{i,k,cu,\mathbb{I}} = (br_{i,k,cu,\mathbb{I}} - I_{i,k,u_0,\mathbb{I}}) / I_{i,k,w,\mathbb{I}} \\ \delta_{i,k,cv,\mathbb{I}} = (br_{i,k,cv,\mathbb{I}} - I_{i,k,v_0,\mathbb{I}}) / I_{i,k,h,\mathbb{I}} \\ \lambda_{i,k,w,\mathbb{I}} = br_{i,k,w,\mathbb{I}} / W \\ \lambda_{i,k,h,\mathbb{I}} = br_{i,k,h,\mathbb{I}} / H \end{cases} \quad (6.19)$$

Inversement, lors de l'inférence les décalages et déformations sont utilisés pour recalculer les coordonnées du rectangle encadrant prédit :

$$\begin{cases} br_{i,k,cu,\mathbb{I}} = \delta_{i,k,cu,\mathbb{I}} \times I_{i,k,w,\mathbb{I}} + I_{i,k,u_0,\mathbb{I}} \\ br_{i,k,cv,\mathbb{I}} = \delta_{i,k,cv,\mathbb{I}} \times I_{i,k,h,\mathbb{I}} + I_{i,k,v_0,\mathbb{I}} \\ br_{i,k,w,\mathbb{I}} = \lambda_{i,k,w,\mathbb{I}} \times W \\ br_{i,k,h,\mathbb{I}} = \lambda_{i,k,h,\mathbb{I}} \times H \end{cases} \quad (6.20)$$

À partir de la deuxième version de YOLO, des changements sont apportés pour introduire des ancres. Les coordonnées du centre du rectangle sont toujours inférées par rapport au coin supérieur gauche de la cellule et sont toujours normalisées par les dimensions de la cellule. Leurs valeurs sont donc comprises entre 0 et 1 et une fonction Sigmoid est utilisée pour contraindre la sortie du réseau dans cette plage de valeurs. Les dimensions du rectangle ne sont plus prédies par rapport aux dimensions de l'image mais proportionnellement à des ancres, comme pour Faster R-CNN. La formule utilisée pour calculer les étiquettes de YOLO V2 est la suivante :

$$\begin{cases} \delta_{i,k,cu,\mathbb{I}} = (br_{i,k,cu,\mathbb{I}} - I_{i,k,u_0,\mathbb{I}}) / I_{i,k,w,\mathbb{I}} \\ \delta_{i,k,cv,\mathbb{I}} = (br_{i,k,cv,\mathbb{I}} - I_{i,k,v_0,\mathbb{I}}) / I_{i,k,h,\mathbb{I}} \\ \lambda_{i,k,w,\mathbb{I}} = \log(br_{i,k,w,\mathbb{I}} / \ddagger_{k,w,\mathbb{I}}) \\ \lambda_{i,k,h,\mathbb{I}} = \log(br_{i,k,h,\mathbb{I}} / \ddagger_{k,h,\mathbb{I}}) \end{cases} \quad (6.21)$$

Contrairement à Faster R-CNN, les ancres ne sont pas choisies manuellement. Elles sont choisies en groupant les rectangles des images d'entraînement selon leurs ratios d'aspect à l'aide de l'algorithme des K moyens (K mean).

Pour retrouver les coordonnées en pixels d'un rectangle grâce aux prédictions de YOLO V2, on utilise la formule suivante :

$$\begin{cases} br_{i,k,cu,\mathbb{I}} = \delta_{i,k,cu,\mathbb{I}} \times I_{i,k,w,\mathbb{I}} + I_{i,k,u_0,\mathbb{I}} \\ br_{i,k,cv,\mathbb{I}} = \delta_{i,k,cv,\mathbb{I}} \times I_{i,k,h,\mathbb{I}} + I_{i,k,v_0,\mathbb{I}} \\ br_{i,k,w,\mathbb{I}} = \ddagger_{k,w,\mathbb{I}} \times \exp(\lambda_{i,k,w,\mathbb{I}}) \\ br_{i,k,h,\mathbb{I}} = \ddagger_{k,h,\mathbb{I}} \times \exp(\lambda_{i,k,h,\mathbb{I}}) \end{cases} \quad (6.22)$$

Maintenant que nous avons vu les rectangles encadrants et leur utilisation pour la localisation d'objets, nous allons présenter les masques de segmentation.

6.4.2 Masque de segmentation et d'instances

La segmentation d'image est plus complexe. Il faut étiqueter chaque pixel pour préciser à quel objet il appartient. Comme pour la classification d'image, le problème des instances d'objets multiples se pose.

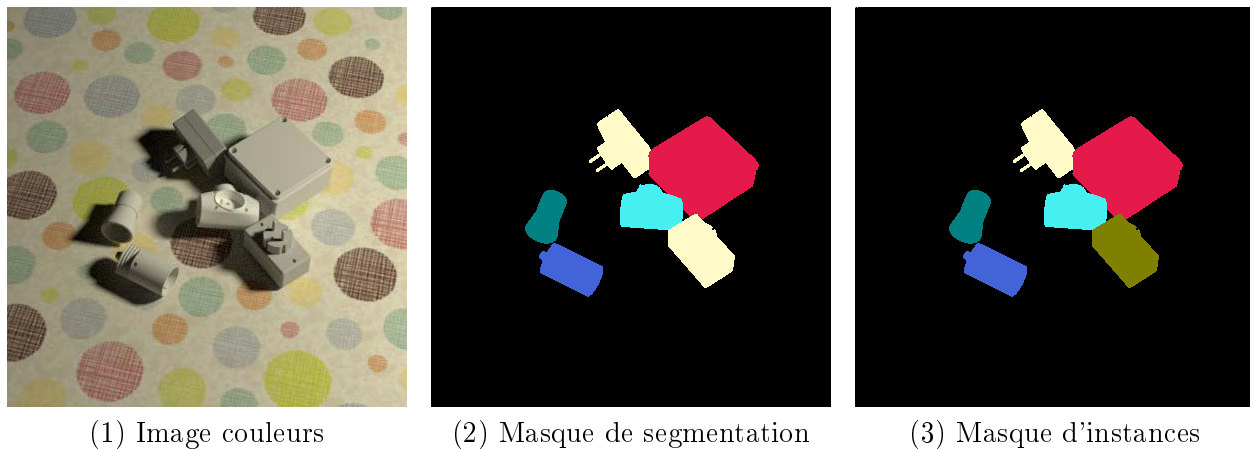


FIGURE 6.8 – Exemple d’une image, du masque de segmentation et du masque des instances du jeu de données T-LESS Synthétique. Chaque catégorie d’objet se voit attribuer une couleur unique dans le masque de segmentation. Dans le masque d’instances, chaque instance d’objet se voit attribuer une couleur. Remarquez les multiplises qui ont la même couleur dans le masque de segmentation mais pas dans le masque d’instances.

Masque de segmentation

Le masque de segmentation associe à chaque pixel un vecteur logique comme pour la classification d’images. L’arrière-plan est aussi considéré comme une catégorie à part entière. Cela permet d’éviter que la majeure partie des étiquettes ne soit nulle, et donc que le gradient soit nul aussi. S’il n’y a qu’une seule instance d’une seule catégorie d’objet, alors le masque de segmentation est similaire au masque vu ci-dessus. De façon générale, on peut le calculer à partir du masque de $B_{i,j}$ de chaque objet $j \in [0, |M|]$:

$$S_{i,u,v,j} = \begin{cases} vrai & \text{si } \widehat{B_{i,j,u,v}} \text{ est vrai} \\ faux & \text{sinon} \end{cases} \quad \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (6.23)$$

Le masque de segmentation peut aussi être stocké comme une image en niveaux de gris où chaque pixel a pour valeur l’index de la catégorie d’objet :

$$S_{i,u,v} = \begin{cases} j + 1 & \text{si } \widehat{B_{i,j,u,v}} \text{ est vrai} \\ 0 & \text{sinon} \end{cases} \quad \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (6.24)$$

On incrémente de un chaque index pour ne pas confondre la première catégorie d’objet d’indice zero et l’arrière-plan.

Masque d’instances

Étiqueter deux instances adjacentes d’un même objet avec le masque de segmentation n’est pas possible, car il n’existe aucune distinction entre les deux. En utilisant des nombres entiers, il est possible d’attribuer une étiquette unique à chaque instance. De cette façon, il est aussi possible de stocker les masques d’instances sous la forme d’images comme le montre la figure 6.8. Mais cette représentation ne peut pas être formellement définie à l’aide d’un vecteur logique par pixel, car le vecteur aurait une longueur variable car il n’y a pas toujours le même nombre d’objets dans une image. Il n’est donc pas possible de prédire directement des masque d’instance avec un réseau de neurones. Pour contourner, on utilise des masques de segmentation et on s’assure qu’il y ait une frontière d’un pixel au moins entre les objets. Cette frontière est définie comme appartenant à l’arrière-plan.

6.5 Étiquettes pour la détection 3D

Cette section présente les étiquettes pour la détection 3D qui sont fournies communément dans les jeux de données de détection 3D. La détection 3D est aussi appelée estimation de la pose ou estimation des six degrés de liberté d'un objet. Son objectif est de trouver, à partir d'une image, la translation et la rotation de chaque objet vu à l'image. Les méthodes doivent donc estimer les trois coefficients pour la translation et les trois angles d'Euler pour la rotation. Certaines estiment aussi dans le même temps les catégories des objets. Avec les réseaux de neurones, il est possible de prédire directement la translation et les angles d'Euler avec par exemple PoseNet++. D'autres méthodes estiment d'abord des points dans le plan image dont elles connaissent les positions dans le repère objet et retrouvent ensuite les paramètres extrinsèques grâce à l'algorithme PnP. Récemment, les embeddings ont permis d'associer densément tous les pixels de l'objet à des coordonnées dans le repère objet (c.f. sous-section 4.3.5, ou ci dessous). Ce qui permet à d'appliquer PnP sur plus de couples 2D-3D et donc de conforter son résultat.

6.5.1 Transformations affines, matrices de rotation et vecteurs de translation

Comme expliqué dans la problématique de cette thèse, la transformation affine $T_{i,j,\mathbb{M} \rightarrow \mathbb{C}}$ est l'opération mathématique qui transforme les points de l'objet en points dans le repère de la caméra. La transformation affine est formulée comme une matrice de dimensions 4×4 . Elle travaille sur des points en coordonnées homogènes (x, y, z, w) . La transformation affine est la composition des opérations de rotation, translation, mise à l'échelle (zoom) et de transvection (shear). Dans notre cas, seule la translation et la rotation sont utilisées, et on considère des objets d'échelle unique, et rigides.

La pose de l'objet $M_{i,j}$ dans l'image I_i peut donc être étiquetée avec la matrice de transformation affine $T_{i,j}$:

$$T_{i,j} = \begin{bmatrix} R_{i,j,0,0} & R_{i,j,0,1} & R_{i,j,0,2} & t_{i,j,0} \\ R_{i,j,1,0} & R_{i,j,1,1} & R_{i,j,1,2} & t_{i,j,1} \\ R_{i,j,2,0} & R_{i,j,2,1} & R_{i,j,2,2} & t_{i,j,1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.25)$$

Lorsque cette matrice est sauvegardée au format textuel ou binaire, la dernière ligne est souvent ignorée, mais elle est néanmoins nécessaire dans les calculs. La rotation correspond alors à la partie gauche et la translation à la dernière colonne de droite. La rotation est sans unité et la translation est exprimée en mètres comme les modèles 3D et les cartes de profondeur. Ces deux sous-parties peuvent aussi être extraites et sauvegardées indépendamment :

$$R_{i,j} = \begin{bmatrix} R_{i,j,0,0} & R_{i,j,0,1} & R_{i,j,0,2} \\ R_{i,j,1,0} & R_{i,j,1,1} & R_{i,j,1,2} \\ R_{i,j,2,0} & R_{i,j,2,1} & R_{i,j,2,2} \end{bmatrix} \quad (6.26)$$

$$t_{i,j} = \begin{bmatrix} t_{i,j,0} \\ t_{i,j,1} \\ t_{i,j,1} \end{bmatrix} \quad (6.27)$$

6.5.2 Angles d'Euler

Une matrice encode la rotation sur neuf valeurs. Les angles d'Euler sont une représentation plus compacte sur trois coefficients, car les axes de rotation sont définis implicitement.

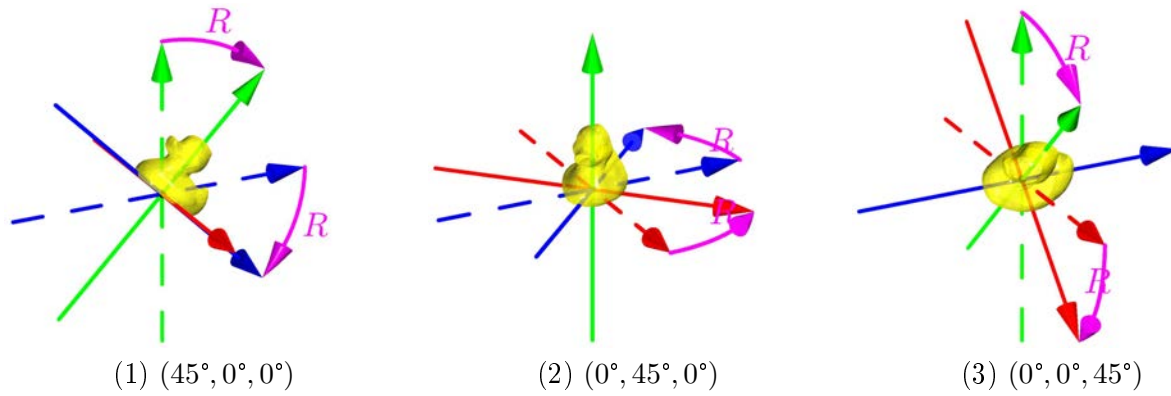


FIGURE 6.9 – Illustration de la rotation avec une matrice de rotation ou un angle d'euler.

La figure 6.9 illustre la rotation avec les angles d'Euler. Les formules suivantes [26, 123] permettent de calculer les trois angles d'Euler en radians à partir d'une matrice de rotation 3D R . Premièrement, il faut calculer la trace de R , car si celle-ci est égale à 0 le calcul est différent (pour éviter une division par 0).

$$\text{trace}(R) = \sqrt{R_{0,0} \times R_{0,0} + R_{1,0} \times R_{1,0}} \quad (6.28)$$

Puis, si la $\text{trace}(R)$ est strictement positive alors les trois angles d'Euler (e_0, e_1, e_2) sont calculés comme :

$$\begin{aligned} e_0 &= \text{atan2}(R_{2,1}, R_{2,2}) \\ e_1 &= \text{atan2}(-R_{2,0}, \text{trace}) \\ e_2 &= \text{atan2}(R_{1,0}, R_{0,0}) \end{aligned} \quad (6.29)$$

Sinon il sont calculés comme :

$$\begin{aligned} e_0 &= \text{atan2}(-R_{1,2}, R_{1,1}) \\ e_1 &= \text{atan2}(-R_{2,0}, \text{trace}) \\ e_2 &= 0 \end{aligned} \quad (6.30)$$

Ce calcul est implémenté dans la bibliothèque *transforms3d*¹[11, 26, 123] et dans la bibliothèque *OpenCV*²[84, 38].

Plusieurs angles d'Euler peuvent correspondre à la même matrice de rotation. De plus, le calcul de la distance entre deux angles d'Euler ne donne pas le même résultat selon le sens de rotation. Par exemple, la distance entre les angles 359° et 1° peut être égale à 358° et 2° . Pour finir, il n'est pas possible d'appliquer la rotation sur des points directement avec des angles d'Euler, il faut d'abord les convertir en une matrice de rotation. Pour ce faire on peut calculer les matrices de rotation sur X, Y, Z par les angles e_0, e_1, e_2 et les multiplier pour obtenir la matrice de rotation sur les trois axes :

$$\begin{aligned} R &= R_Z(e_2) \times R_Y(e_1) \times R_X(e_0) \\ &= \begin{bmatrix} \cos(e_2) & -\sin(e_2) & 0 \\ \sin(e_2) & \cos(e_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(e_1) & -\sin(e_1) \\ 0 & \sin(e_1) & \cos(e_1) \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(e_0) & -\sin(e_0) \\ 0 & \sin(e_0) & \cos(e_0) \end{bmatrix} \end{aligned} \quad (6.31)$$

Note : la multiplication de matrices se fait par la droite. La matrice de rotation sur X est donc à droite, car c'est celle qui est appliquée en première. Celle sur Y est au milieu, car elle intervient en deuxième position. Puis, à gauche, c'est celle sur l'axe Z qui intervient en dernière. Si les matrices sont multipliées dans un autre ordre, alors le résultat obtenu ne sera pas le même du fait de la non commutativité de la multiplication de matrices ($A \times B \neq B \times A$).

1. `transforms3d.euler.mat2euler()`
2. `cv2.Rodrigues()`

6.5.3 Quaternion

Un quaternion $q = (x, y, z, w)$ est une autre représentation mathématique de la rotation, très utilisée en 3D. Les quaternions offrent les mêmes avantages de calcul que les matrices de rotation mais sont plus compactes : 4 coefficients au lieu de 9. Les quaternions sont des nombres complexes exprimés sans unité.

Pour trouver le quaternion q , il faut calculer la matrice Q [68] :

$$Q = \frac{1}{3} \begin{bmatrix} R_{0,0} - R_{1,1} - R_{2,2} & 0 & 0 & 0 \\ R_{0,1} + R_{1,0} & R_{1,1} - R_{0,0} - R_{2,2} & 0 & 0 \\ R_{0,2} + R_{2,0} & R_{1,2} + R_{2,1} & R_{2,2} - R_{0,0} - R_{1,1} & 0 \\ R_{2,0} - R_{1,2} & R_{0,2} - R_{2,0} & R_{1,0} - R_{0,1} & R_{0,0} + R_{1,1} + R_{2,2} \end{bmatrix} \quad (6.32)$$

On calcule ensuite les vecteurs et valeurs propres de la matrice Q :

$$\Lambda, V = \text{eigh}(Q) \quad (6.33)$$

Le quaternion $q = (x, y, z, w)$ est alors égal au vecteur propre V_i associé à la plus grande valeur propre Λ_i de la matrice K :

$$q = x, y, z, w = V_i \mid \forall j \in [0, n[, i \neq j, \Lambda_i \geq \Lambda_j \quad (6.34)$$

De plus, on préférera les quaternions avec un poids w positif :

$$q = -q \text{ si } w < 0 \text{ sinon } q \quad (6.35)$$

Ce calcul est implémenté dans la bibliothèque *transforms3d*³[11, 26, 125, 68]

Pour calculer la rotation d'un point p avec un quaternion, la formule suivante peut être utilisée :

$$f(p) = q \cdot p \cdot q^{-1} \quad (6.36)$$

Et pour retrouver la matrice de rotation 3D R à partir du quaternion q on peut utiliser la formule suivante[68] :

$$R = \begin{bmatrix} 2(q_0q_0 + q_1q_1) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 2(q_0q_0 + q_2q_2) - 1 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 2(q_0q_0 + q_3q_3) - 1 \end{bmatrix} \quad (6.37)$$

Pour une explication détaillée et graphique de ce que sont les quaternions et de leur fonctionnement, nous vous recommandons fortement cette série de vidéos interactives réalisées par Grant Sanderson et Ben Eater[103, 2, 1] : <https://eater.net/quaternions/>

6.5.4 Correspondances 2D-3D

PnP [76, 71] permet de calculer la rotation et la translation à partir de correspondances 2D-3D. Pour les correspondances, on utilise des points dont on connaît la position dans le repère objet et qui seront détectés dans l'image. Formellement, si on définit un ensemble de points $P_{i,j,k,\mathbb{M}} \forall k$ définis dans le repère objet et, que l'on a détecté ces points en 2D dans l'image $P_{i,j,k,\mathbb{I}} \forall k$, alors étant donné la matrice de paramètres intrinsèques K , PnP nous donne une estimation de la matrice de rotation \widehat{R} et du vecteur de translation \widehat{t} tel que :

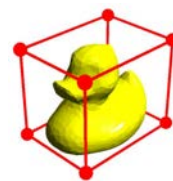
$$\widehat{R}_{i,j}, \widehat{t}_{i,j} = \text{PnP}(P_{j,\mathbb{M}}, P_{i,j,\mathbb{I}}, K) \quad (6.38)$$

$$P_{i,j,\mathbb{I}} \simeq P_{\mathbb{C} \rightarrow \mathbb{I}} \times \left[\widehat{R}_{i,j} | \widehat{t}_{i,j} \right] \times P_{j,\mathbb{M}} \quad (6.39)$$

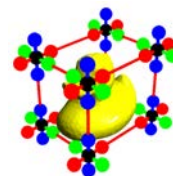
3. `transforms3d.quaternions.mat2quat()`



(1) Boîte encadrante visualisé en superposition de l'image. BB8 [91] utilise des boîtes sous cette forme comme prédictions.



(2) Boîte encadrante visualisé dans le repère objet.



(3) Boîte encadrante avec plus de points pour chaque coin visualisé dans le repère objet. Elle est utilisée sous cette forme par la méthode ayant inspiré BB8 [21].

FIGURE 6.10 – Exemple de la boîte encadrante du Canard de LINEMOD.

N'importe quel ensemble de points peut être utilisé. La bibliothèque OpenCV utilise les coins d'un plateau de dame pour la calibration caméra, car ceux-ci sont facilement identifiables et localisables dans une image. On peut utiliser les points de tout le modèle d'un objet ou un sous-ensemble. On peut utiliser les points au bout des parties saillantes d'un objet, en espérant qu'ils soient visuellement plus discernables. On peut aussi utiliser des points quelconques sans rapport avec l'apparence de l'objet, comme les coins de la boîte encadrante ou les extrémités des axes du repère objet.

Les coordonnées des points sur l'image peuvent être utilisés directement comme étiquette. Il est aussi possible de passer par des représentations intermédiaires comme des cartes donnant la direction vers les points.

Ci-dessous nous présentons deux méthodes. La première prédit directement les coordonnées des coins de la boîte encadrante. La deuxième prédit des cartes donnant la direction vers sept des points les plus éloignés du modèle de l'objet.

BB8 [91] utilise les huit coins de la boîte encadrante de l'objet. Un exemple de boîte encadrante dans le repère objet et dans l'image est présenté à la figure 6.10. La boîte encadrante $bb_{j,\mathbb{M}}$ est tout d'abord calculée dans le repère de l'objet. Ses huit coins sont calculés à partir des sommets du modèle 3D M_j de l'objet de la façon suivante :

$$\left\{ \begin{array}{l}
 bb_{j,0,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,1,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,2,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,3,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,4,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,5,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,6,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmin}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,z,\mathbb{M}} \right) \\
 bb_{j,7,\mathbb{M}} = \left(\underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,x,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,y,\mathbb{M}}, \underset{k \in [0, |M_j|]}{\operatorname{argmax}} M_{j,k,z,\mathbb{M}} \right)
 \end{array} \right. \quad (6.40)$$

Il est ensuite possible de calculer la position de cette boîte englobante dans le repère 3D de la caméra et dans le plan image, grâce aux formules définies dans la problématique :

$$bb_{i,j,\mathbb{C}} = [R_{i,j,\mathbb{M} \rightarrow \mathbb{C}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{C}}] \times bb_{j,\mathbb{M}} \quad (6.41)$$

$$bb_{i,j,\mathbb{I}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times [R_{i,j,\mathbb{M} \rightarrow \mathbb{S}} | t_{i,j,\mathbb{M} \rightarrow \mathbb{S}}] \times bb_{j,\mathbb{M}} \quad (6.42)$$

$$(6.43)$$

Ces points peuvent être utilisés comme étiquettes pour entraîner un réseau de neurones à les prédire. On retrouvera la matrice de rotation et le vecteur de translation avec l'algorithme PnP comme expliqué ci-dessus.

Il est aussi possible d'utiliser plus de points pour chaque coin de la boîte encadrante comme ce fut proposé par [21]. Par exemple sept points par coin : le coin et six points supplémentaires pris en se décalant dans une des six directions du repère 3D (à gauche, à droite, en avant, en arrière, au-dessus et en dessous). Cette représentation est illustrée à la figure 6.10.

6.5.5 Correspondances 2D-3D Denses

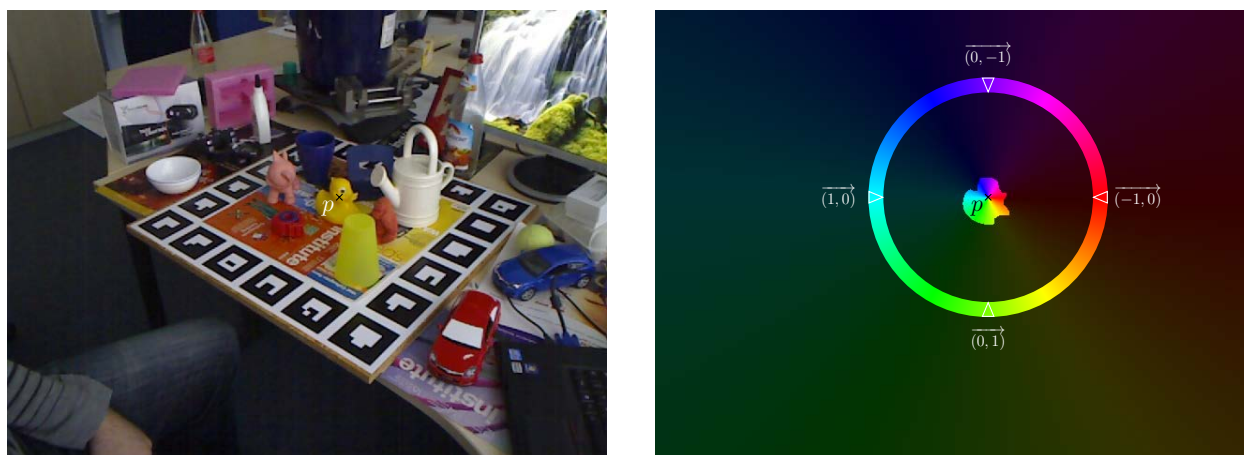
La méthode PV-Net [86] utilise un nombre de points choisis les plus espacés possibles à la surface du modèle 3D de l'objet. Les points $P_{i,j}$ sont projetés sur l'image à l'aide de la formule suivante :

$$P_{i,j,\mathbb{I}} = P_{\mathbb{C} \rightarrow \mathbb{I}} \times [\widehat{R}_{i,j} | \widehat{t}_{i,j}] \times P_{j,\mathbb{M}} \quad (6.44)$$

Il est possible de prédire directement les coordonnées des points, mais les auteurs de PV-Net ont proposé une autre approche. Pour chaque point, une carte de vecteurs est construite. Les pixels de ces cartes sont des vecteurs qui pointent vers le point associé à la carte, et de magnitude unitaire. Un exemple est présenté à la figure 6.11. Pour sept points, il y aura donc sept cartes. Formellement, la carte $D_{i,j,k}$ associée au point $P_{i,j,k}$ est calculé comme :

$$D_{i,j,k,u,v} = \frac{\overrightarrow{(u,v) - P_{i,j,k}}}{\|\overrightarrow{(u,v) - P_{i,j,k}}\|_2} \quad \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (6.45)$$

Lors de l'inférence, le réseau prédit ces cartes plutôt que des points. Le réseau utilise une architecture de type encodeur-décodeur type U-Net [97] avec des connexions résiduelles



(1) Image couleur.

 (2) Carte de directions vers le point p .

FIGURE 6.11 – Illustration des étiquettes de PVNet [86]. Pour chaque point considéré par PVNet, une carte de vecteur de même dimensions que l'image est calculée. Chaque pixel de cette carte contient un vecteur à 2 coefficients qui donne la direction du pixel vers le point. Il est de magnitude unitaire.



(1) Projection cylindrique.

(2) Projection sphérique.

FIGURE 6.12 – Illustration de l'embedding cylindrique et sphérique du Canard de LINEMOD. Les deux méthodes donnent des résultats similaires.

[48]. Cette architecture et cette représentation des étiquettes permettent d'associer à chaque pixel de l'image couleur, la direction vers les points à trouver. Toutes ces prédictions sont ensuite prises en compte grâce à un algorithme RANSAC qui va calculer les coordonnées les plus probables étant donné le champ de directions.

6.5.6 Embedding 2D-3D

PV-Net infère en chaque pixel les directions vers les points considérés. Comme nous l'avons vu, cette représentation mathématique peut être visualisée avec des couleurs. Le nombre de correspondances 2D-3D reste limité à sept points, même si celles-ci sont plus robustes aux occlusions du fait de la contribution de chaque pixel.

Les auteurs de DPOD [129] ont poussé cette idée plus loin. Ils associent indirectement chaque pixel de l'objet visible à des coordonnées 3D. Prédire directement des coordonnées 3D est compliqué, du fait de l'absence de l'information de profondeur. C'est pourquoi les auteurs de DPOD ont d'abord associé à chaque coordonnée 3D à la surface du modèle de l'objet une couleur unique à deux composantes : u et v . Pour réaliser cette association, il est possible de texturer l'objet dans Blender ou Meshlab comme le montre la figure 6.12. Le réseau est ensuite utilisé pour prédire ces couleurs en chaque pixel de l'image. On obtient alors autant de correspondances 2D-3D que de pixels de l'objet visible. Il est aussi nécessaire que le réseau prédise le masque de segmentation de l'objet pour ne pas inclure les pixels de

l'arrière-plan dans les calculs de PnP.

Cette technique a aussi été étendue pour prédire des couleurs représentatives de la géométrie locale des objets ou de sous parties des objets [90]. Grâce à cette méthode, le réseau peut généraliser et prédire la pose d'objet qu'il n'a pas connu lors de son entraînement.

6.5.7 Conclusion

Le succès et la puissance des réseaux de neurones a d'abord laissé penser qu'ils seraient capables de résoudre d'eux-mêmes le problème de l'ambiguïté de la profondeur due à la projection perspective. Mais prédire directement la translation et la rotation en trois dimensions ne fonctionne pas aussi bien que de prédire des correspondances 2D-3D et d'utiliser l'algorithme PnP. PnP fait un très bon travail pour résoudre le problème de projection perspective. Ne pas l'utiliser et essayer de l'apprendre donne plus de travail au CNN. De plus, les étiquettes utilisées pour établir les correspondances 2D-3D ont évolué pour devenir robustes aux occlusions puis généralisées à plusieurs objets.

Dans le prochain chapitre nous allons présenter les mesures et fonctions de coût que l'on calcule avec ces étiquettes.

Chapitre 7

Mesures et fonctions de coûts

Introduction

Dans ce chapitre nous présentons les mesures de précisions et les fonctions de coûts pour les trois tâches que nous considérons. Nous commencerons par la classification d'images, puis la détection d'objets et enfin l'estimation des six degrés de liberté des objets. Chaque tâche utilise des étiquettes différentes et les fonctions utilisées sont donc différentes. Pour la classification d'image nous présenterons par exemple la crossentropie catégorique. Dans le cas de la localisation d'objets, nous verrons comment mesurer la distances entre deux rectangle encadrant, mais aussi l'intersection sur l'union comme mesure de recalage entre deux rectangles. Et enfin, pour l'estimation des six degrés de liberté, nous présenterons comment calculer l'erreur de recalage 3D. Pour les trois tâches, nous verrons aussi comment sont définis les résultats positifs, négatifs, vrais et faux et leurs combinaisons. Ce qui nous permettra à terme de calculer la précision, le rappel ou la précision moyenne, quelle que soit la tâche.

7.1 Mesures et coûts pour la détection 1D

7.1.1 Mesures

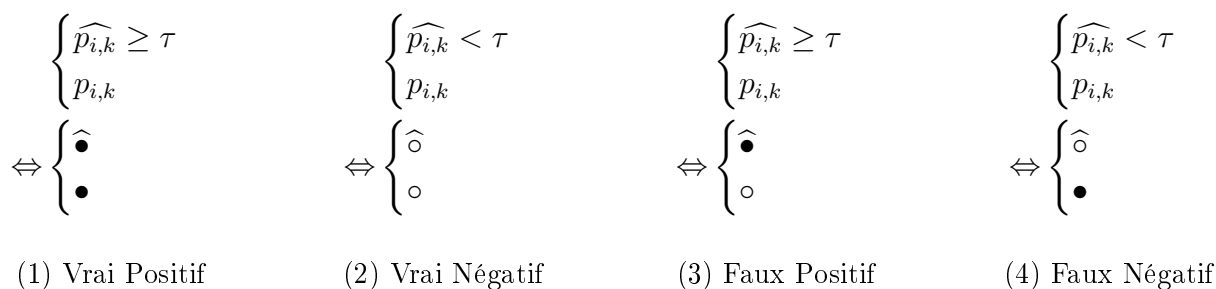


FIGURE 7.1 – Illustration des différents cas de figures entre prédictions et étiquettes.

Positifs et Négatifs

Un réseau de neurones pour la détection 1D ne prend pas de "décision" mais donne un avis pour chaque catégorie d'objet considérée. Pour une image I_i , le résultat est un vecteur de probabilités, noté \widehat{p}_i . Chaque valeur $\widehat{p}_{i,k}$ de ce vecteur est comprise entre 0 et 1. Une forte

probabilité $\widehat{p}_{i,k}$ indiquera la présence d'un objet k à l'image I_i , et inversement une faible probabilité indiquera son absence.

Les annotations correspondantes sont, elles aussi, formulées comme un vecteur, mais à valeurs booléennes : objet présent \bullet , ou objet absent soit \circ , i.e. 0 ou 1, mais rien entre les deux.

Pour pouvoir comparer prédiction et étiquettes, il faut convertir les prédictions dans le même format que les étiquettes. Plusieurs techniques sont possibles. Si on connaît le nombre d'objets présents dans les images, on peut prendre les t prédictions avec les plus fortes probabilités. Cette technique est par exemple utilisée pour calculer l'erreur top-1 ou top-5. Si on ne connaît pas le nombre d'objets à l'image, on peut utiliser un seuil de décision. La valeur de seuil utilisée, notée τ , tranchera entre objet présent ou absent, i.e. détection positive ou négative en fonction de la probabilité donnée par le réseau :

$$\begin{cases} \text{positif}(\widehat{p}_{i,k}, \tau) & \text{si } \widehat{p}_{i,k} \geq \tau \\ \text{négatif}(\widehat{p}_{i,k}, \tau) & \text{sinon} \end{cases} \quad (7.1)$$

C'est le programmeur qui, en définissant ce seuil, définit la sensibilité du réseau et conditionne la prise de décision finale. C'est ici la limite de l'IA dont les médias font la gloire. Le résultat de l'IA dans le cas de la détection 1D n'est qu'une probabilité et la décision finale revient encore à l'être humain qui fixe le seuil. Mais comme nous allons le voir dans la section consacrée aux résultats, les probabilités obtenues sont très discriminantes et robustes avec les réseaux de neurones. De plus, les mesures de performance servent à évaluer numériquement les qualités et les défauts d'un réseau pour un jeu de données choisi. La valeur du seuil τ peut donc être optimalement choisie grâce aux mesures que nous allons présenter ici.

On notera de la même façon une étiquette positive ou négative :

$$\begin{cases} \text{positif}(p_{i,k}) & \text{si } p_{i,k} \\ \text{négatif}(p_{i,k}) & \text{sinon} \end{cases} \quad (7.2)$$

Vrai et Faux

Maintenant que prédictions et étiquettes sont dans le même format, nous allons pouvoir les comparer pour identifier les bons et mauvais résultats. Ce qui va nous permettre de définir les mesures que nous utiliserons. Premièrement, on peut définir deux cas de figures :

1. Si le **résultat correspond à l'étiquette**, alors il est **vrai**.
2. Si le **résultat ne correspond pas à l'étiquette**, alors il est **faux**.

La sous-section "Vrai et Faux" donne une mesure pour ces deux cas de base.

Rentrons maintenant dans les détails. Nous comparons deux valeurs : une prédiction et une étiquette. Chacune peut avoir deux valeurs possibles : objet présent, objet absent. On a donc $2^2 = 4$ cas de figures :

1. Si le résultat est **positif** et **correspond à l'étiquette**, alors c'est un **vrai positif**.
2. Si le résultat est **négatif** et **correspond à l'étiquette**, alors c'est un **vrai négatif**.
3. Si le résultat est **positif** mais **ne correspond pas à l'étiquette**, alors c'est un **faux positif**.
4. Le résultat est **négatif** mais **ne correspond pas à l'étiquette**, alors c'est un **faux négatif**.

La figure 7.1 illustre ces quatre cas. Chaque cas de figure nous apprend des choses différentes sur les réseaux que nous allons évaluer. Ci-dessous nous détaillons chacun des cas et ce qu'il nous apprend.

Les vraies et fausses prédictions sont identifiables grâce aux formules suivantes :

$$\begin{cases} \text{vrai}(\widehat{p}_{i,k}, p_{i,k}, \tau) \text{ si} & (\text{positif}(\widehat{p}_{i,k}, \tau) \wedge \text{positif}(p_{i,k})) \\ & \vee (\text{négatif}(\widehat{p}_{i,k}, \tau) \wedge \text{négatif}(p_{i,k})) \\ \text{faux}(\widehat{p}_{i,k}, p_{i,k}, \tau) \text{ sinon} \end{cases} \quad (7.3)$$

Les vraies prédictions sont de bonnes prédictions et les fausses de mauvaises. Plus le nombre de vraies prédictions est grand, meilleur est le réseau étudié. De façon opposée, plus le nombre de fausses prédictions est grand, pire est le réseau.

Moyenne de vrais ou de faux

Pour mesurer la proportion de résultats vrais, on calcule généralement la "*Mean Average Precision*" (en anglais). C'est la mesure de performance la plus utilisée dans la littérature. Le nom de cette mesure se traduit très mal en français : "*Moyenne Moyenne Précision*" ? Il s'agit en effet d'une double moyenne. La première se calcule sur les $|M|$ valeurs du vecteur \widehat{p}_i . La deuxième se calcule sur les $|I|$ images considérées pendant la phase de test. C'est donc une double moyenne sur l'ensemble des objets et sur l'ensemble des images de test.

Pour une valeur de seuil τ , la proportion des résultats vrais se calcule comme :

$$\text{MAP}(\widehat{p}_i, p_i, \tau) = \frac{1}{|I|} \frac{1}{|M|} \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si vrai}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.4)$$

Dans le cas parfait ou toutes les prédictions sont vraies, la mesure sera égale à 1. Plus les résultats de détections seront faux, moins la valeur mesurée sera grande. Jusqu'à ce que tous les résultats soient faux auquel cas la mesure vaudra 0.

La proportion de faux résultats se calcule de la même façon :

$$\text{MAE}(\widehat{p}_i, p_i, \tau) = \frac{1}{|I|} \frac{1}{|M|} \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si faux}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.5)$$

Elle est égale à $1 - \text{MAP}$ pour une même valeur de τ . Il est intéressant de prendre une valeur élevée pour τ pour calculer MAP. Par exemple, nous utilisons 0.75. En effet, on souhaite que les objets présents dans les images soient classifiés comme présents par le réseau de la façon la plus franche possible. On souhaite donc que le réseau ait une forte confiance et donc une probabilité $\widehat{p}_{i,k}$ élevée.

De façon plus large et plus objective, on calcule aussi cette mesure avec différentes valeurs de τ . Le résultat obtenu est alors une courbe présentant les valeurs de MAP en fonction des valeurs de τ . Les deux axes de ce graphique étant compris entre 0 et 1, on peut ensuite calculer l'aire sous cette courbe, dont la valeur sera aussi comprise entre 0 et 1.

Cette mesure nous apprendra donc si le réseau fonctionne bien. Nous allons maintenant rentrer dans les détails des quatre cas possibles qui nous apprennent des choses différentes.

Vrais Positifs

Un vrai positif est identifié grâce à la formule suivante :

$$\text{vrais positifs}(\widehat{p}_{i,k}, p_{i,k}, \tau) \Leftrightarrow \text{positif}(\widehat{p}_{i,k}, \tau) \wedge \text{positif}(p_{i,k}) \quad (7.6)$$

Le nombre de vrais positifs (TP) est calculé sur l'ensemble des images et des objets visibles dans ces images. Pour évaluer la proportion de vrais positifs (MTP), on divise ce résultat par le nombre d'étiquettes positives :

$$TP(\widehat{p}, p, \tau) = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si vrais positifs}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.7)$$

$$Z^+ = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si positif}(p_{i,k}) \\ 0 & \text{sinon} \end{cases} \quad (7.8)$$

$$MTP(\widehat{p}, p, \tau) = \frac{1}{Z^+} TP(\widehat{p}, p, \tau) \quad (7.9)$$

On va donc calculer la proportion d'objets identifiés comme présents à juste titre. Comme pour la MAP, il est intéressant de choisir une valeur de τ élevée. Cela confirmera que ces objets sont identifiés avec une grande confiance par le réseau. On peut aussi calculer l'air sous la courbe pour plusieurs valeurs de τ .

Vrais Négatifs

Un vrai négatif est identifié grâce à la formule suivante :

$$\text{vrais négatifs}(\widehat{p}_{i,k}, p_{i,k}, \tau) \Leftrightarrow \text{négatif}(\widehat{p}_{i,k}, \tau) \wedge \text{négatif}(p_{i,k}) \quad (7.10)$$

Pour évaluer la proportion de vrais négatifs (MTN), on va calculer le nombre de vrais négatifs prédits (TN). De la même manière que précédemment, on va comparer cette valeur au nombre d'étiquettes négatives. De cette façon, la mesure vaudra 1 si tous les objets non-présents sont bien détectés comme tel. Et sa valeur diminuera avec chaque prédiction injustement positive. On a donc :

$$TN(\widehat{p}, p, \tau) = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si vrai négatif}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.11)$$

$$Z^- = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si négatif}(p_{i,k}) \\ 0 & \text{sinon} \end{cases} \quad (7.12)$$

$$MTN(\widehat{p}, p, \tau) = \frac{1}{Z^-} TN(\widehat{p}, p, \tau) \quad (7.13)$$

Pour une même valeur de τ cette mesure calcule l'opposé des vrais positifs. Mais ici, il est intéressant de choisir une valeur de τ plus petite. Nous utilisons 0.25. De cette façon, cette mesure nous indiquera si les objets identifiés, comme absents à juste titre, le sont avec une probabilité nettement plus faible. Ce qui est un résultat souhaitable, car les objets présents et absents seront ainsi mieux séparés.

Si on se fixe deux seuils τ_{\ominus} et τ_{\oplus} , on pourra identifier les objets dont la probabilité inférieure à τ_{\ominus} comme clairement négatifs, ceux dont la probabilité est supérieur à τ_{\oplus} comme clairement positifs, et ceux dont la probabilité est comprise entre ces deux seuils comme indécis.

Faux Positifs

Les faux positifs sont le premier cas de mauvais résultats. Ils sont identifiés grâce à la formule suivante :

$$\text{faux positifs}(\widehat{p}_{i,k}, p_{i,k}, \tau) \Leftrightarrow \text{positif}(\widehat{p}_{i,k}, \tau) \wedge \text{négatif}(p_{i,k}) \quad (7.14)$$

Pour calculer la proportion de faux positifs (MFP), il faut d'abord compter le nombre de faux positifs (FP), puis le diviser par le nombre d'étiquettes négatives. Ce premier résultat nous donnera une valeur comprise en 0, s'il n'y a pas de faux positifs, et 1, si tous les objets absents sont prédits comme présents. Il suffit donc de prendre l'opposé de cette valeur par rapport à 1 :

$$\text{FP}(\widehat{p}, p, \tau) = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si faux positif}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.15)$$

$$\text{MFP}(\widehat{p}, p, \tau) = 1 - \frac{1}{Z^-} \text{FP}(\widehat{p}, p, \tau) \quad (7.16)$$

Pour calculer les faux positifs, il est important d'utiliser le même seuil que pour les vrais positifs. Nous avons choisi 0.75. On veut évidemment que cette mesure soit la meilleure possible, mais que signifie-t-elle ? Prenons des exemples. Dans le cas d'une application de suggestion de contenus, comme Netflix qui vous suggère des films, cela reviendrait à ce que l'application vous propose des films qui ne vous plairont pas. Ce qui est problématique. Pour une voiture autonome, si elle perçoit des obstacles ou des panneaux de limitation de vitesses qui n'existent pas, elle s'arrêtera ou roulera moins vite, ce qui impacte le trajet, mais pas la sécurité des passages ou des usagés de la route. C'est aussi problématique, mais ne constitue pas un problème de sécurité des personnes. Les faux négatifs sont pires dans ce cas pratique, comme nous allons le voir ci-dessous. Pour notre application de réalité mixte d'aide à la maintenance, des objets inexistant reconnus pourraient se retrouver augmentés visuellement alors qu'ils ne sont pas là. Ou l'application pourrait perdre le déroulé du scénario de maintenance.

Faux Négatifs

Les faux négatifs sont le deuxième cas de mauvais résultats. Ils sont identifiés grâce à la formule suivante :

$$\text{faux négatifs}(\widehat{p}_{i,k}, p_{i,k}, \tau) \Leftrightarrow \text{négatif}(\widehat{p}_{i,k}, \tau) \wedge \text{positif}(p_{i,k}) \quad (7.17)$$

Pour calculer la proportion de faux négatifs (MFN), comme pour les faux positifs, nous allons d'abord calculer la proportion de faux négatifs par rapport au nombre d'étiquettes positives. Puis nous soustrairons ce résultat à 1 pour obtenir une mesure entre 0 et 1 et non l'inverse.

$$\text{FN}(\widehat{p}, p, \tau) = \sum_{i=0}^{|I|} \sum_{k=0}^{|M|} \begin{cases} 1 & \text{si faux négatif}(\widehat{p}_{i,k}, p_{i,k}, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.18)$$

$$\text{FN}(\widehat{p}, p, \tau) = 1 - \frac{1}{Z^+} \text{FN}(\widehat{p}, p, \tau) \quad (7.19)$$

On utilisera aussi le même seuil que le pour les vrais négatifs. Pour nous, c'est 0.25.

Les faux négatifs sont beaucoup plus critiques pour les applications. Reprenons l'exemple d'une voiture autonome. Si elle ne reconnaît pas le piéton qui se trouve sur un passage clouté alors elle risque de le renverser. Il s'agit d'une faute grave qui met en danger la vie des personnes. Dans le cas d'un site de streaming comme Netflix, ce dernier ne vous proposera pas certains films susceptibles de vous plaire, et donc vous proposera à la place des films qui vous plairont moins. Cela peut donc être aussi problématique que les faux positifs. Pour notre application de réalité mixte d'aide à la maintenance, l'application ne détectera pas les objets sur lesquels travaille l'utilisateur, ils ne seront donc pas augmentés et l'utilisateur ne sera pas guidé. Ce qui revient à dire que l'application ne fonctionnera pas.

Précision et Rappel

Maintenant que nous avons les quatre cas possibles et les mesures associées, nous allons voir qu'il est possible de calculer deux mesures qui combinent ces résultats. Ces mesures sont la précision et le rappel [133, 33].

La précision est le rapport entre le nombre de vrais positifs et le nombre de positifs. Le nombre de positifs est égal à la somme des vrais et faux positifs. La précision vaudra donc 1 s'il n'y a que des vrais positifs. Si le nombre de vrais positifs diminue, ou si le nombre de faux positifs augmente, alors la valeur de la précision diminuera.

$$\text{précision}(\hat{p}, p, \tau) = \frac{\text{TP}(\hat{p}, p, \tau)}{\text{TP}(\hat{p}, p, \tau) + \text{FP}(\hat{p}, p, \tau)} \quad (7.20)$$

Le rappel est le rapport entre le nombre de vrais positifs et la somme des vrais positifs et des faux négatifs. La valeur de cette mesure décroît si le nombre de vrais positifs diminuent ou si le nombre de faux négatifs augmente.

$$\text{rappel}(\hat{p}, p, \tau) = \frac{\text{TP}(\hat{p}, p, \tau)}{\text{TP}(\hat{p}, p, \tau) + \text{FN}(\hat{p}, p, \tau)} \quad (7.21)$$

7.1.2 Fonctions de coûts

Une fonction de coût est utilisée pour entraîner un réseau. Sa valeur est calculée sur chaque paquet d'exemples d'entraînements. On cherche à la minimiser lors de la descente de gradient. Sa valeur doit donc être grande si les prédictions sont éloignées des étiquettes, décroître lorsque les prédictions se rapprochent des étiquettes, et être nulle si les prédictions sont identiques aux étiquettes.

Pour nos expériences de détection 1D, i.e. de classification d'images, les prédictions et étiquettes sont des vecteurs de probabilités. Une valeur de probabilité pour chaque catégorie d'objet. Dans ces expériences nous utilisons la fonction de cross-entropie catégorique [81].

$$\text{CCE}(\hat{p}_i, p_i) = - \sum_{k=0}^{|\mathcal{M}|} p_{i,k} \times \log(\hat{p}_{i,k}) \quad (7.22)$$

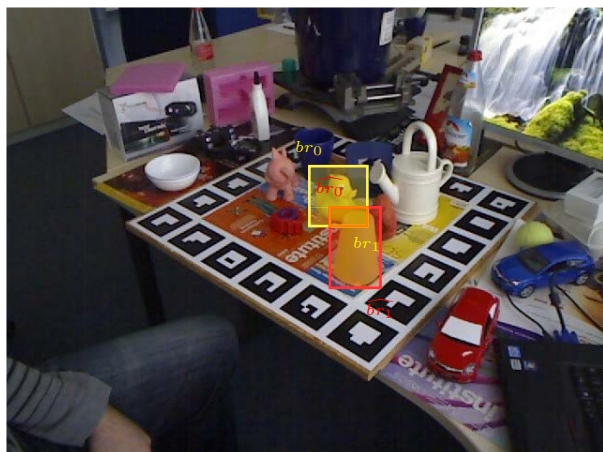
Cette fonction multiplie la probabilité étiquette et le logarithme de la probabilité prédite (pour chaque objet k).

La probabilité étiquette est strictement égale à 0 ou 1. Les objets non-présent dans les étiquettes ne contribuent donc pas à la fonction de coût. Du fait du produit, les faux positifs ne contribuent donc pas non plus.

La probabilité prédite prend des valeurs comprises entre 0 et 1 (grâce à l'activation de la dernière couche dense, le "*softmax*"). Si la probabilité prédite est égale à 1, alors son logarithme vaudra 0 et le produit avec l'étiquette vaudra 0. La "distance" entre prédiction et étiquette sera donc bien nulle. Mais plus la probabilité prédite est proche de 0 (faux négatif), plus la valeur de son logarithme tendra vers $-\infty$ et le produit avec la probabilité étiquette aussi. On aura donc une forte "distance" (à valeur négative) entre prédiction et étiquette.

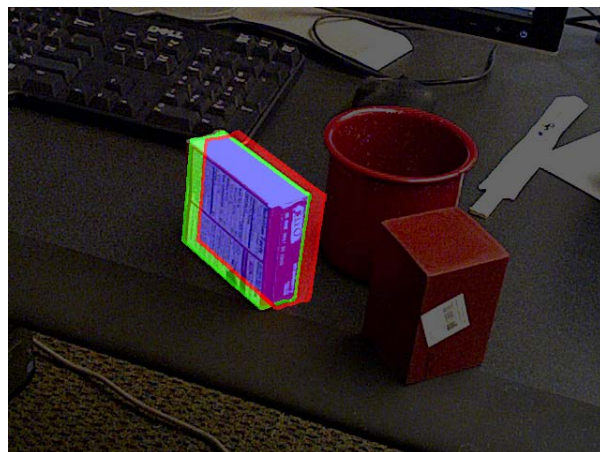
Le $-$ devant l'équation vient "redresser" les valeurs négatives en valeurs positives pour que celles-ci conviennent à des distances qui seront minimisées lors de la descente de gradient.

Lors de l'entraînement, les exemples d'entraînement sont considérés par paquets ("*batch*" en anglais). Pour calculer le gradient, le coût est calculé sur tous les exemples d'un paquet. Pour chaque expérience, nous présentons aussi le coût total calculé sur tous les exemples d'entraînement, donc toutes les images et leurs étiquettes :



(1) Illustration du calcul de l'intersection sur l'union :

$$IoU(\text{□}, \text{□}) = \text{□} / (\text{□} + \text{□} - \text{□})$$



(2) Illustration du calcul de l'intersection sur l'union avec des masques de segmentation.

$$IoU(\text{□}, \text{□}) = \text{□} / (\text{□} + \text{□} - \text{□})$$

FIGURE 7.2 – Calcul de l'intersection sur l'union entre deux rectangles encadrants ou entre deux masques de segmentations.

$$\mathcal{L}_{\text{class}} = \sum_{i=0}^{|I|} CCE(\hat{p}_i, p_i) \quad (7.23)$$

D'autres fonctions de coûts robustes aux erreurs d'annotation ont aussi été proposées comme la moyenne absolue des erreurs [43] et la cross-entropie généralisée [132].

Conclusion

Nous avons donc à notre disposition six mesures permettant d'analyser les résultats d'un réseau de neurones pour la classification d'images, et plusieurs fonctions de coûts dont la très célèbre cross-entropie catégorique. Dans la prochaine section, nous allons présenter les mesures et fonctions de coûts pour la détection 2D.

7.2 Mesures et coûts pour la détection 2D

7.2.1 Mesures

Intersection sur L'Union : IoU

Les méthodes de détection 2D cherche à résoudre deux tâches : 1. localiser les objets et; 2. les identifier. Ces deux tâches sont évaluées séparément dans la littérature. Nous avons vu au chapitre précédent comment mesurer l'efficacité d'une méthode pour la tâche de classification des images. Ces mêmes mesures sont utilisées pour évaluer la capacité d'une méthode à correctement identifier les objets. Pour la localisation, on a besoin d'une technique qui mesure la coïncidence entre la zone de l'image, Z qui est associée à un objet par la méthode de détection 2D (la prédiction), par rapport à la zone que cet objet occupe réellement dans l'image (l'étiquette). Que la zone soit formulée comme un rectangle encadrant ou un masque de segmentation, c'est généralement la méthode de l'intersection sur l'union qui est utilisée, i.e. l'IoU. Elle est définie comme le ratio d'aire de la zone de coïncidence de la prédiction et de l'étiquette divisée par l'aire jointe de la prédiction et de l'étiquette. L'IoU peut se calculer

sur les rectangles encadrants aussi bien que sur les masques de segmentation comme nous allons le voir dans les deux sous-sections suivantes.

IoU sur les rectangle encadrants On rappelle qu'un rectangle encadrant br est défini comme un tableau de deux points (le coin supérieur gauche identifié par l'indice 0 et inférieur droit par l'indice 1) :

$$br_{i,k,\mathbb{I}} = \begin{bmatrix} br_{i,k,0,u,\mathbb{I}} & br_{i,k,0,v,\mathbb{I}} \\ br_{i,k,1,u,\mathbb{I}} & br_{i,k,1,v,\mathbb{I}} \end{bmatrix} \quad (7.24)$$

Et que la largeur, la hauteur et l'aire d'un rectangle sont calculés comme :

$$largeur(br_{i,k,\mathbb{I}}) = br_{i,k,1,u,\mathbb{I}} - br_{i,k,0,u,\mathbb{I}} \quad (7.25)$$

$$hauteur(br_{i,k,\mathbb{I}}) = br_{i,k,1,v,\mathbb{I}} - br_{i,k,0,v,\mathbb{I}} \quad (7.26)$$

$$aire(br_{i,k,\mathbb{I}}) = largeur(br_{i,k,\mathbb{I}}) \times hauteur(br_{i,k,\mathbb{I}}) \quad (7.27)$$

Les rectangles encadrants sont très utilisés en partie pour la facilité avec laquelle l'IoU est calculée avec des rectangles. C'est aussi une des formes les plus simples pour délimiter une zone d'image. Si on considère deux rectangles br (étiquette) et \widehat{br} (prédiction) pour calculer l'IoU, on commence par calculer l'intersection, qui est un rectangle :

$$intersection(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}}) = \begin{bmatrix} \max(br_{i,k,0,u,\mathbb{I}}, \widehat{br}_{i,k,0,u,\mathbb{I}}) & \max(br_{i,k,0,v,\mathbb{I}}, \widehat{br}_{i,k,0,v,\mathbb{I}}) \\ \min(br_{i,k,1,u,\mathbb{I}}, \widehat{br}_{i,k,1,u,\mathbb{I}}) & \min(br_{i,k,1,v,\mathbb{I}}, \widehat{br}_{i,k,1,v,\mathbb{I}}) \end{bmatrix} \quad (7.28)$$

L'union est plus compliquée à calculer car c'est un rectangle uniquement si les deux rectangles coïncident parfaitement. Dans la plupart des cas, il s'agit d'un polygone, voire de deux rectangles distincts. Heureusement, il n'est pas nécessaire de la calculer sa forme puisque tout ce dont on a besoin est de son aire. Or, on peut la calculer très facilement à partir de l'aire des deux rectangles et de l'aire de l'intersection des deux :

$$\begin{aligned} aire(union(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}})) &= aire(br_{i,k,\mathbb{I}}) \\ &+ aire(\widehat{br}_{i,k,\mathbb{I}}) \\ &- aire(intersection(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}})) \end{aligned} \quad (7.29)$$

Pour finir l'IoU est donc calculé comme :

$$IoU(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}}) = \frac{aire(intersection(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}}))}{aire(union(br_{i,k,\mathbb{I}}, \widehat{br}_{i,k,\mathbb{I}}))} \quad (7.30)$$

La figure 7.21 présente un exemple visuel.

IoU sur les masques de segmentation Les pixels d'un masque de segmentation peuvent être considérés comme des carrés de côté 1 et donc d'aire 1. L'aire de l'objet segmenté dans un masque de segmentation se calcule en comptant les pixels égaux à 1 (*vrai*). L'intersection entre deux masques se calcule par une simple application de l'opération "et logique" pixels à pixels. De même pour l'union qui se calcule par application de l'opération "ou logique"

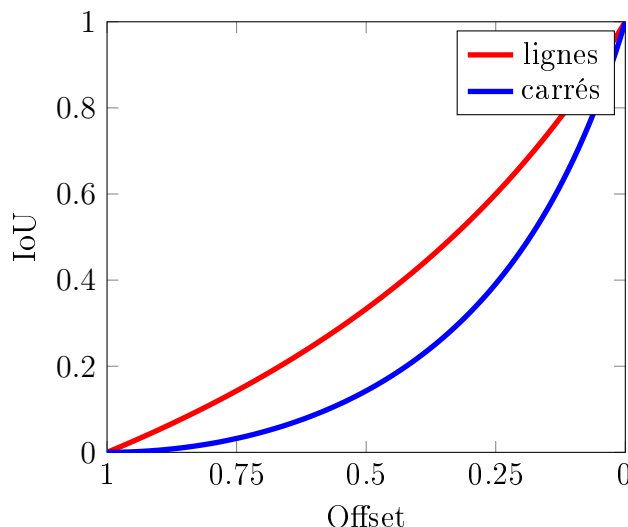





FIGURE 7.3 – Illustration du changement de l’IoU par rapport au décalage entre deux lignes et deux rectangles. Un décalage horizontal et vertical d’un rectangle introduit des changements d’ordre quadratique d’IoU puisque cette dernière est exprimée en terme d’aire. Lorsque l’IoU est calculée sur des lignes alors le changement est plus linéaire mais garde une tendance quadratique. — Un écart de 1 indique que les deux rectangles ou les deux lignes sont complètement dissociés  — Un écart de 0.5 indique que les rectangles sont décalés de 0.5 en largeur et hauteur (soit $\simeq 0.7$ en diagonale). Pour les lignes l’écart est seulement horizontal).  — Un écart de 0 indique que les deux rectangles se superposent parfaitement. Même chose pour les lignes. 

pixels à pixels. On peut donc calculer l’IoU entre deux masques de segmentation :

$$aire(B_{i,\mathbb{B}}) = \sum_{(u,v) \in |\mathbb{B}|^2} \begin{cases} 1 & \text{if } B_{i,u,v,\mathbb{B}} = \text{vrai} \\ 0 & \text{else} \end{cases} \quad (7.31)$$

$$intersection(B_{i,\mathbb{B}}, \widehat{B_{i,\mathbb{B}}}) = \{ B_{i,u,v,\mathbb{B}} \wedge \widehat{B_{i,u,v,\mathbb{B}}} \quad \forall (u,v) \in |\mathbb{B}|^2 \} \quad (7.32)$$

$$union(B_{i,\mathbb{B}}, \widehat{B_{i,\mathbb{B}}}) = \{ B_{i,u,v,\mathbb{B}} \vee \widehat{B_{i,u,v,\mathbb{B}}} \quad \forall (u,v) \in |\mathbb{B}|^2 \} \quad (7.33)$$

$$IoU(B_{i,\mathbb{B}}, \widehat{B_{i,\mathbb{B}}}) = \frac{aire(intersection(B_{i,\mathbb{B}}, \widehat{B_{i,\mathbb{B}}}))}{aire(union(B_{i,\mathbb{B}}, \widehat{B_{i,\mathbb{B}}}))} \quad (7.34)$$

$$(7.35)$$

Remarques L’intersection sur l’union appliquée à des rectangles est exprimée en terme de rapport d’aires. Les aires sont exprimées en carré : m^2 , cm^2 , mm^2 , etc... puisqu’il s’agit du produit de deux longueurs. De ce fait, l’IoU sur des rectangles suit une fonction quadratique entre 0 et 1 dans sa réponse. Comme on peut le voir à la figure 7.3, une IoU égale à 0.6 indique en réalité que deux rectangles se superposent très bien avec moins de 0.125 de décalage pour des rectangles de côté 1. En effet l’IoU croît moins vite que l’écart entre deux rectangles ne se réduit. Il en est de même pour l’IoU calculée sur des masques de segmentation, i.e. des pixels, car ceux-ci sont carrés par définition. Il est important de prendre en compte ce facteur dans l’interprétation des résultats présentés sous forme d’IoU. De même si l’IoU est utilisée lors de l’entraînement d’une méthode comme fonction de coût cette propriété impactera l’entraînement.

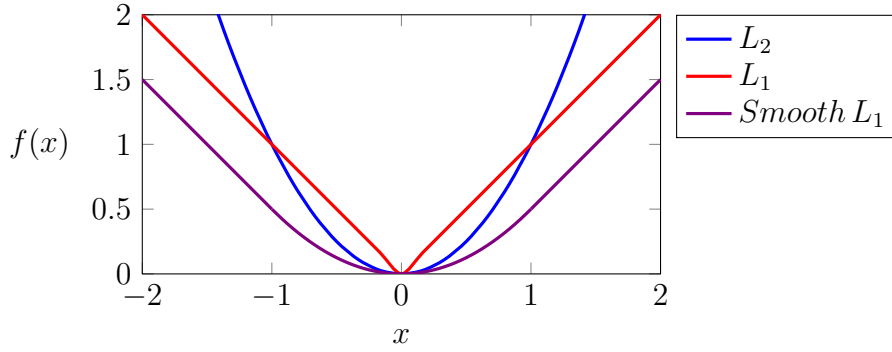


FIGURE 7.4 – Illustration de plusieurs fonctions de coûts.

Vrai et Faux

Les notions de prédictions vrais et fausses par rapport à une étiquette peuvent aussi être formulées pour la localisation d'objet grâce à l'IoU et un seuil manuel τ . Pour un rectangle encadrant prédit $\widehat{br}_{i,k}$ et un rectangle encadrant étiquette $br_{i,k}$:

$$\begin{cases} \text{vrai}(\widehat{br}_{i,k}, br_{i,l}, \tau) & \text{si } IoU(\widehat{br}_{i,k}, br_{i,l}) \geq \tau \\ \text{faux}(\widehat{br}_{i,k}, br_{i,l}, \tau) & \text{sinon} \end{cases} \quad (7.36)$$

Il est aussi possible de définir les notions de vrai positif, faux négatif et faux positif :

- Un vrai positif est une prédiction qui coïncide bien avec une étiquette.
- Un faux positif est une prédiction qui ne coïncide pas bien avec une étiquette.
- Un faux négatif est une étiquette pour avec laquelle aucune prédiction ne coïncide suffisamment.

Ce qui donne sous forme d'équation :

$$\text{vrai positif}(\widehat{br}_{i,k}, br_i, \tau) \Leftrightarrow \exists l \mid \text{vrai}(\widehat{br}_{i,k}, br_{i,l}, \tau) \quad (7.37)$$

$$\text{faux positif}(\widehat{br}_{i,k}, br_i, \tau) \Leftrightarrow \nexists l \mid \text{vrai}(\widehat{br}_{i,k}, br_{i,l}, \tau) \quad (7.38)$$

$$\text{faux négatif}(\widehat{br}_i, br_{i,l}, \tau) \Leftrightarrow \nexists k \mid \text{vrai}(\widehat{br}_{i,k}, br_{i,l}, \tau) \quad (7.39)$$

Il n'est pas possible de définir la notion de vrai négatif pour cette tâche. En effet, il n'y a pas d'étiquette pour définir l'absence d'objet du fait que l'arrière-plan soit lui aussi considéré comme une catégorie d'objet.

Précision, Rappel et MAP

Comme pour la classification d'image il est possible de calculer la précision, le rappel, et la MAP à partir des calculs des vrais positifs, faux négatifs et faux positifs :

$$\begin{aligned} \text{précision}(\widehat{p}, p, \tau) &= \frac{TP(\widehat{p}, p, \tau)}{TP(\widehat{p}, p, \tau) + FP(\widehat{p}, p, \tau)} \\ \text{rappel}(\widehat{p}, p, \tau) &= \frac{TP(\widehat{p}, p, \tau)}{TP(\widehat{p}, p, \tau) + FN(\widehat{p}, p, \tau)} \end{aligned}$$

7.2.2 Fonctions de coûts

Le but de la fonction de coût est de mesurer l'écart entre un rectangle encadrant prédit $\widehat{br}_{i,k}$ et l'étiquette associée $br_{i,k}$. La fonction doit renvoyer une valeur nulle quand les rectangles sont identiques et, large quand ils sont fortement dissociés. La première intuition est d'utiliser la distance Euclidienne. Mais cette fonction, qui fait intervenir plusieurs carrés et une racine carrée, est souvent jugée trop coûteuse en terme de calcul. Nous allons voir ici les principales fonctions de calcul utilisées par les méthodes de localisation d'objets existantes.

Faster R-CNN

Faster R-CNN est le premier réseau qui s'est abstrait de la recherche sélective pour prédire les rectangles encadrants directement avec un réseau de neurones. Il faut donc entraîner le réseau RPN pour cette tâche et donc exprimer une fonction de coût qui permettra à la descente de gradient de fonctionner. Les auteurs ont choisi de mesurer l'erreur entre un rectangle encadrant prédit $\widehat{br}_{i,k}$ et l'étiquette associée $br_{i,k}$ grâce à la distance L_1 lissée (ℓ_1) :

$$\mathcal{L}_{\text{loc}}(\widehat{br}_{i,k}, br_{i,k}) = \sum_{c \in \{u,v,w,h\}} \ell_1(br_{i,k,c} - \widehat{br}_{i,k,c}) \quad (7.40)$$

La norme L1 lissée est calculée comme suit et la réponse de cette fonction est illustrée à la figure 7.4.

$$\ell_1(x) = \begin{cases} 0.5x^2 & \text{si } |x| > 1 \\ |x| - 0.5 & \text{sinon} \end{cases} \quad (7.41)$$

YOLO

YOLO utilise une fonction de distance Euclidienne sans racine carrée. Les erreurs de positionnements (calculés sur les coordonnées u et v) et de tailles (calculés sur la largeur w et la hauteur h) des rectangles sont additionnées séparément. Les tailles des rectangles sont traitées séparément car le réseau prédit la racine carrée de la largeur et de la hauteur au lieu de les prédire directement. Cela permet à de faibles erreurs dans des grands rectangles de moins contribuer à la fonction de coût.

$$\mathcal{L}_{\text{loc}}(\widehat{br}_{i,k}, br_{i,k}) = \sum_{c \in \{u,v\}} (br_{i,k,c} - \widehat{br}_{i,k,c})^2 + \sum_{c \in \{w,h\}} (br_{i,k,c} - \sqrt{\widehat{br}_{i,k,c}})^2 \quad (7.42)$$

Conclusion

Dans cette section, nous avons vu qu'il est possible de définir les mêmes mesures pour la détection 2D que pour la détection 3D. Les fonctions de coûts utilisées sont inspirées des mathématiques appliquées à la géométrie des rectangles, modulo quelques raccourcis de calculs pour alléger les calculs lors de l'entraînement.

La prochaine section présente les mesures pour la détection 3D.

7.3 Mesures et coûts pour la détection 3D

Les méthodes de détection 3D ou d'estimation des six degrés de liberté sont divisées en deux groupes. Celles de premier groupe prédisent directement les six degrés de liberté : translation et rotation. Celles du second prédisent des ensembles de points sur l'image qui

permettent avec PnP de retrouver la translation et la rotation. Dans les deux cas, on obtient la pose estimée des objets $\{\widehat{R}_{i,k}, \widehat{t}_{i,k}\} \forall k$. Nous avons aussi à notre disposition un ensemble de poses étiquettes $\{R_{i,j}, t_{i,j}\} \forall j$ comme mentionné au chapitre précédent.

Dans cette section, nous allons détailler le calcul des mesures et des fonctions de coûts utilisées dans la littérature. Les mesures ont encore une fois pour but de caractériser en moyenne la qualité des résultats de détection par rapport aux étiquettes. Alors que les fonctions de coût mesurent l'écart entre résultats et étiquettes pour la descente de gradient.

7.3.1 Mesures

Mesures sur les paramètres de poses

L'estimation des six degrés de liberté est avant tout un problème mathématique où l'on cherche deux vecteurs : un vecteur de position, i.e. la translation exprimée en mètres, et un vecteur d'orientation, i.e. la rotation, sous la forme d'angles d'Euler en degrés ou en radians. La première façon de mesurer la qualité d'une prédiction par rapport à une étiquette est donc de mesurer la distance entre les vecteurs. Cette distance est ensuite comparée à un seuil de décision qui tranche entre vrais et faux résultats.

Vrais et faux Pour la translation on peut mesurer l'erreur indépendamment sur X , Y et Z ou bien mesurer l'erreur diagonale :

$$\text{vrai}_x(\widehat{t}_{i,k}, t_{i,l}, \tau) \Leftrightarrow |\widehat{t}_{i,k,x} - t_{i,l,x}| < \tau \quad (7.43)$$

$$\text{vrai}_y(\widehat{t}_{i,k}, t_{i,l}, \tau) \Leftrightarrow |\widehat{t}_{i,k,y} - t_{i,l,y}| < \tau \quad (7.44)$$

$$\text{vrai}_z(\widehat{t}_{i,k}, t_{i,l}, \tau) \Leftrightarrow |\widehat{t}_{i,k,z} - t_{i,l,z}| < \tau \quad (7.45)$$

$$\text{vrai}_{x,y,z}(\widehat{t}_{i,k}, t_{i,l}, \tau) \Leftrightarrow \|\widehat{t}_{i,k} - t_{i,l}\|_2 < \tau \quad (7.46)$$

$$\Leftrightarrow \sqrt{\sum_{c \in \{x,y,z\}} (\widehat{t}_{i,k,c} - t_{i,l,c})^2} < \tau \quad (7.47)$$

$$\Leftrightarrow \sum_{c \in \{x,y,z\}} (\widehat{t}_{i,k,c} - t_{i,l,c})^2 < \tau^2 \quad (7.48)$$

Même chose pour la rotation avec des angles d'Euler prédis $\widehat{e}_{i,k}$ et étiquettes $e_{i,l}$:

$$\text{vrai}_\alpha(\widehat{e}_{i,k}, e_{i,l}, \tau) \Leftrightarrow \mathcal{D}_{Euler}(\widehat{e}_{i,k,\alpha}, e_{i,l,\alpha}) < \tau \quad (7.49)$$

$$\text{vrai}_\beta(\widehat{e}_{i,k}, e_{i,l}, \tau) \Leftrightarrow \mathcal{D}_{Euler}(\widehat{e}_{i,k,\beta}, e_{i,l,\beta}) < \tau \quad (7.50)$$

$$\text{vrai}_\gamma(\widehat{e}_{i,k}, e_{i,l}, \tau) \Leftrightarrow \mathcal{D}_{Euler}(\widehat{e}_{i,k,\gamma}, e_{i,l,\gamma}) < \tau \quad (7.51)$$

$$\text{vrai}_{\alpha,\beta,\gamma}(\widehat{e}_{i,k}, e_{i,l}, \tau) \Leftrightarrow \sqrt{\sum_{c \in \{\alpha,\beta,\gamma\}} \mathcal{D}_{Euler}(\widehat{e}_{i,k,c}, e_{i,l,c})^2} < \tau \quad (7.52)$$

Sauf que la distance entre deux angles d'Euler (compris entre 0 et 2π) doit être calculée dans les deux sens de rotation pour trouver la distance correcte entre les deux angles, qui est la plus petite des deux :

$$\mathcal{D}_{Euler}(a, b) = \min(a - b, b - a) \quad (7.53)$$

Finalement pour qu'une prédiction de pose comprenant rotation et translation soit correcte, il faut que la translation et la rotation soient correctes :

$$\text{vrai}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_{i,l}, e_{i,l}, \tau_m, \tau_r) \Leftrightarrow \text{vrai}_{x,y,z}(\widehat{t}_{i,k}, \widehat{e}_{i,l}, \tau_m) \wedge \text{vrai}_{\alpha,\beta,\gamma}(\widehat{e}_{i,k}, \widehat{e}_{i,l}, \tau_r) \quad (7.54)$$

Vrais positifs, faux positifs, faux négatifs Les vrais positifs, faux positifs et faux négatifs sont ensuite définissables comme pour la localisation d'objets.

$$\text{vrai positif}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_i, e_i, \tau_m, \tau) \Leftrightarrow \exists l \mid \text{vrai}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_{i,l}, e_{i,l}, \tau_m, \tau) \quad (7.55)$$

$$\text{faux positif}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_i, e_i, \tau_m, \tau) \Leftrightarrow \nexists l \mid \text{vrai}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_{i,l}, e_{i,l}, \tau_m, \tau) \quad (7.56)$$

$$\text{faux négatif}(\widehat{t}_i, \widehat{e}_i, t_{i,l}, e_{i,l}, \tau_m, \tau) \Leftrightarrow \nexists k \mid \text{vrai}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_{i,l}, e_{i,l}, \tau_m, \tau) \quad (7.57)$$

Cinq centimètres et cinq degrés La définition des vrais et faux résultats que nous avons vue ci-dessus a donné une mesure connue sous le nom de $5cm5^\circ$. Comme son nom l'indique, un couple translation et rotation est considéré comme correct si la translation est vraie à un seuil de $5cm$ et si la rotation est vraie à un seuil de 5° . Autrement la prédiction est considérée comme fausse. Avec cette définition des vrais et des faux, la MAP7.1.1 est ensuite calculée comme d'habitude pour obtenir le résultat :

$$5cm5^\circ(\widehat{t}, \widehat{e}, t, e) = \frac{1}{|I|} \frac{1}{|K|} \sum_{i=0}^{|I|} \sum_{k=0}^{|K|} \begin{cases} 1 & \text{si vrai positif}(\widehat{t}_{i,k}, \widehat{e}_{i,k}, t_i, e_i, 0.05m, 5^\circ) \\ 0 & \text{sinon} \end{cases} \quad (7.58)$$

Mesures de recalages 3D

L'estimation des six degrés de liberté d'un objet peut aussi être vue comme un problème de recalage où l'on cherche à aligner le modèle 3D sur l'objet réel. Le même procédé peut être défini grâce aux étiquettes. Dans ce cas, on cherche à aligner le modèle 3D de l'objet sur le même modèle défini dans le repère de la caméra. Si la pose prédite est identique à la pose étiquette, les deux modèles seront parfaitement alignés. Sinon on pourra mesurer une distance entre les points des deux modèles.

Vrais et faux

$$\mathcal{D}_{\text{mean}}^{3D}(M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}) = \frac{1}{|K|} \sum_{k=0}^{|K|} \left\| \left[\widehat{R}_{i,j} | \widehat{t}_{i,j} \right] M_{i,j,k} - [R_{i,j} | t_{i,j}] M_{i,j,k} \right\|_2 \quad (7.59)$$

$$\text{vrai}(M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}, \tau) \Leftrightarrow \mathcal{D}_{\text{mean}}^{3D}(M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}) < \tau \quad (7.60)$$

La distance de recalage moyenne $\mathcal{D}_{\text{mean}}^{3D}$ n'est pas bornée, car plus la prédiction de translation est distante de l'étiquette, plus les modèles seront éloignés. Elle dépend aussi de la taille de l'objet. Si on tourne un très petit objet sur lui-même, l'erreur mesurée sera petite aussi, même si l'angle est grand alors. Par contre, si on tourne un grand objet sur lui-même, alors l'erreur sera grande, même si l'angle est petit. Le seuil τ (en mètres) utilisé dépend donc de chaque objet. Pour le recalage 3D, on utilise généralement un seuil égal à 10 % du diamètre de l'objet :

$$\tau = 0.1 \text{ diamètre}(M_{i,j}) \quad (7.61)$$

$$\text{diamètre}(M_{i,j}) = 2 \operatorname{argmax}_{k \in [0, K]} \|M_{i,j,k}\|_2 \quad (7.62)$$

On a donc :

$$\text{vrai}(M_{i,j}, \widehat{t}_{i,j}, \widehat{R}_{i,j}, t_{i,j}, R_{i,j}) \Leftrightarrow \mathcal{D}_{\text{mean}}^{3D}(M_{i,j}, \widehat{t}_{i,j}, \widehat{R}_{i,j}, t_{i,j}, R_{i,j}) < 0.1 \text{ diamètre}(M_{i,j}) \quad (7.63)$$

Vrais positifs, faux positifs, faux négatifs Les vrais positifs, faux positifs et faux négatifs sont identifiés avec les mêmes équations que pour la mesure $5cm5^\circ$ (7.55) où l'on remplacera la fonction "vrai" par celle de l'ADD 3D ci-dessus.

ADD 3D L'ADD 3D qui mesure la proportion d'objets correctement recalés en 3D est finalement calculée comme la moyenne des vrais positifs :

$$\text{ADD}^{3D}(\widehat{R}, \widehat{t}, R, t) = \frac{1}{|I|} \frac{1}{|J|} \sum_{i=0}^{|I|} \sum_{j=0}^{|J|} \begin{cases} 1 & \text{si vrai positif}(M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_i, t_i) \\ 0 & \text{sinon} \end{cases} \quad (7.64)$$

Mesures de recalages 2D

La mesure ADD 3D mesure l'erreur en 3D. Elle est donc particulièrement informative dans le cadre d'applications travaillant en 3D comme la robotique. Dans le cas d'applications de réalité mixte, la même mesure peut être définie sur l'image en 2D. Cette mesure, appelée ADD 2D, nous informe sur le bon alignement du modèle de l'objet projeté à la pose prédite sur l'image, par rapport au même modèle projeté grâce aux étiquettes. Si elle est satisfaite cela signifie que les rendus et visuels, qui seront synthétisés par l'application, apparaîtront correctement alignés sur l'image et que la perspective sera correcte.

Pour la calculer, il suffit de rajouter aux équations précédentes 7.3.1 la projection perspective définie dans la première partie de cette thèse 2.43.

Vrais et faux Un résultat vrai est identifié comme précédemment, mais cette fois-ci, les points sont projetés sur l'image grâce à la matrice de projection perspective 2.43.

$$\mathcal{D}_{\text{mean}}^{2D}(P_i, M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}) = \frac{1}{|K|} \sum_{k=0}^{|K|} \left\| P_i \left[\widehat{R}_{i,j} | \widehat{t}_{i,j} \right] M_{i,j,k} - P_i \left[R_{i,j} | t_{i,j} \right] M_{i,j,k} \right\|_2 \quad (7.65)$$

$$\text{vrai}(P_i, M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}, \tau) \Leftrightarrow \mathcal{D}_{\text{mean}}^{2D}(K_i, M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_{i,j}, t_{i,j}) < \tau \quad (7.66)$$

Vrais positifs, faux positifs, faux négatifs Les vrais positifs, faux positifs et faux négatifs sont identifiés avec les mêmes équations que pour les mesures $5cm5^\circ$ (7.55) et ADD 3D où l'on remplacera la fonction vrai par celle ci-dessus.

ADD 2D L'ADD 2D est finalement calculée comme la moyenne des vrais positifs :

$$\text{ADD}^{2D}(P, \widehat{R}, \widehat{t}, R, t, \tau) = \frac{1}{|I|} \frac{1}{|J|} \sum_{i=0}^{|I|} \sum_{j=0}^{|J|} \begin{cases} 1 & \text{si vrai positif}(P_i, M_{i,j}, \widehat{R}_{i,j}, \widehat{t}_{i,j}, R_i, t_i, \tau) \\ 0 & \text{sinon} \end{cases} \quad (7.67)$$

Le seuil τ communément utilisé est de 5 pixels. Cette mesure peut aussi être appliquée au résultat des réseaux qui estiment les positions de points en 2D. C'est utile par exemple pendant l'entraînement pendant lequel PnP n'est pas appliqué. Cela permet aussi de comparer les réseaux prédisant des points 2D entre eux, sans que PnP n'entre en compte.

7.3.2 Fonctions de Coûts

La fonction de coût utilisée pour entraîner un réseau de neurones dépend du résultat produit par le réseau. Comme nous l'avons vu dans l'état de l'art, il existe principalement deux techniques pour prédire les 6 degrés de liberté d'un objet. Le réseau prédit soit directement les paramètres de poses, ou localise des points en 2D qui, par association avec des points en 3D dans le repère objet et grâce à l'algorithme PnP, vont par la suite permettre de retrouver les paramètres de poses.

Coût sur les paramètres de poses

La plupart des réseaux prédisant directement les paramètres de poses prédisent la translation comme un vecteur. Seul PoseCNN prédit la direction 2D vers le centre de l'objet et une carte de profondeur, ce qui lui permet de calculer la translation après coup. Les méthodes se différencient généralement par la façon dont elles modélisent la rotation. En effet, plusieurs modèles mathématiques existent pour définir la rotation en 3D. La plus connue, que nous utilisons tous les jours, est les angles d'Euler. La matrice de rotation est une forme matricielle des angles, utilisée en algèbre linéaire et dans les moteurs 3D. Moins connus, les quaternions utilisent une représentation basée sur un nombre complexe à quatre coefficients.

Plusieurs critères différencient ces représentations. Les angles d'Euler sont la représentation la plus compacte, car l'axe de rotation est implicite. Mais comme nous l'avons vu, calculer la distance entre deux angles d'Euler impose des considérations. Les matrices de rotations sont utilisables directement dans OpenGL et on peut calculer la différence entre deux matrices facilement. Mais toutes les matrices de transformation 3D ne sont pas des matrices de rotations, la sortie du réseau doit donc être contrainte, ce qui n'est pas trivial et peut déstabiliser le réseau. Les quaternions sont la représentation la moins visuelle, car les calculs sont réalisés en 4D. Mais la distance entre deux quaternions est facile à calculer.

Quand on choisit une représentation des paramètres de pose, deux contraintes prévalent :

1. Est-ce que le réseau pourra apprendre facilement cette représentation ?
2. Est-ce qu'il est facile de formuler la fonction de coût ?

La première contrainte est directement liée aux performances et à la qualité des résultats, comme nous le verrons plus tard. La deuxième est purement de l'ordre de l'implémentation et de la faisabilité. Car implémenter la fonction de coût, surtout sur GPU, n'est pas évident, et ce, même avec l'aide des bibliothèques logicielles modernes.

Vecteur de translation et quaternion Les quaternions sont une représentation très astucieuse de la rotation. Ils facilitent grandement les calculs pour appliquer la rotation sur un ensemble de points, comme nous l'avons vu au chapitre précédent.

Pour mesurer l'erreur entre deux quaternions, et donc le coût de la descente de gradient, on calcule la distance Euclidienne entre les deux quaternions ($\|\widehat{q} - q\|_2$). Mais il faut prendre quelques précautions. Premièrement, les quaternions ne modélisent la rotation que s'ils ont une norme unitaire ($\|q\|_2 = 1$). Il faut donc s'assurer que les étiquettes soient normalisées ($\frac{q_{i,j}}{\|q_{i,j}\|_2}$). Deuxièmement, on ne considère que des quaternions dont le dernier coefficient est positif ($q = -q$ si $q_w < 0$ sinon q). Cette contrainte impose le sens de rotation autour de l'axe Z.

La fonction de coût complète est la suivante :

$$\mathcal{L}(\widehat{q}_{i,j}, \widehat{t}_{i,j}, q_{i,j}, t_{i,j}) = \lambda_q \left\| \widehat{q}_{i,j} - \frac{q_{i,j}}{\|q_{i,j}\|_2} \right\|_2 + \lambda_t \|\widehat{t}_{i,j} - t_{i,j}\|_2 \quad (7.68)$$

Cette fonction de coût est utilisée par exemple par PoseNet. Les hyper-paramètres λ_q et λ_t sont utilisés pour équilibrer les deux termes de la fonction de coût. En effet, la translation n'est pas bornée alors que la distance entre deux quaternions est bornée par $2\sqrt{2}$. C'est pourquoi les auteurs de PoseNet ont utilisé un λ_q représentatif de l'échelle des scènes : compris entre 120 et 750 en intérieur et entre 250 et 2000 en extérieur. Augmenter λ_q leur permet de ramener les valeurs du terme de rotation au même ordre de grandeur que les valeurs du terme consacré à la translation.

Comme nous le verrons plus tard, il est difficile de choisir les valeurs de λ_q et λ_t , car les deux termes de cette fonction de coût peuvent entrer en compétition pendant l'entraînement.

Vecteur de translation et angle d'Euler Les angles d'Euler sont la représentation la plus compacte de la rotation avec 3 coefficients. Une fonction de coût peut être formulée à l'aide de la formule de distance précédemment définie :

$$\mathcal{L}(\widehat{e}_{i,j}, \widehat{t}_{i,j}, e_{i,j}, t_{i,j}) = \lambda_e \mathcal{D}_{\text{Euler}}(\widehat{e}_{i,j}, e_{i,j}) + \lambda_t \|\widehat{t}_{i,j} - t_{i,j}\|_2 \quad (7.69)$$

Les angles d'Euler comme représentation de la rotation avec cette fonction de coût ont été utilisés dans PoseNet++ [130]. Comme pour la représentation avec les quaternions, il faut équilibrer manuellement les deux termes grâce aux hypermètres λ_e et λ_t .

Coût sur des points 2D

PnP est un algorithme qui, à partir de points en 2D sur l'image et leurs équivalents 3D dans le repère objet, permet de calculer les paramètres de poses. Il est donc possible d'utiliser un CNN pour prédire les points sur l'image, pour retrouver ensuite la rotation et la translation. Ce type de CNN est très utilisé pour prédire le rectangle encadrant d'objets dans le cadre de la détection d'objets en 2D.

Points d'une boîte encadrante La boîte encadrante 3D est l'extension naturelle du rectangle encadrant 2D. C'est pour cette raison que c'est l'une des premières représentations à avoir été utilisée pour la détection 3D notamment par BB8 [91]. La fonction de coût de BB8 est la distance Euclidienne entre points prédits et points à la pose étiquette. BB8 prédit les coordonnées des points directement en pixels, mais cette formule fonctionne aussi pour des points dont les coordonnées sont normalisées entre 0 et 1.

$$\mathcal{L}_{\text{loc}}(\widehat{bb}_{i,j}, bb_{i,j}) = \sum_{k=0}^K \|\widehat{bb}_{i,j,k} - bb_{i,j,k}\|_2 \quad (7.70)$$

$$= \sum_{k=0}^K \sqrt{(\widehat{bb}_{i,j,k,u} - bb_{i,j,k,u})^2 + (\widehat{bb}_{i,j,k,v} - bb_{i,j,k,v})^2} \quad (7.71)$$

La racine carrée qui intervient dans le calcul de la distance Euclidienne peut être ignorée pour simplifier les calculs. Cette technique a, par exemple, été utilisée par la méthode [115].

$$\mathcal{L}_{\text{loc}}(\widehat{bb}_{i,j}, bb_{i,j}) = \sum_{k=0}^K (\widehat{bb}_{i,j,k,u} - bb_{i,j,k,u})^2 + (\widehat{bb}_{i,j,k,v} - bb_{i,j,k,v})^2 \quad (7.72)$$

Carte dense de vecteur unitaire de directions Au lieu de prédire directement les coordonnées des points 2D, certaines méthodes comme PVNet [86] et Pose CNN [126] ont choisi de prédire la direction vers ces points en chaque pixel de l'image grâce à un encodeur-décodeur. Les directions vers les points sont modélisées comme des vecteurs unitaires regroupés en carte de mêmes dimensions que l'image. Le coût est calculé comme la somme des différences pixel à pixel. Au lieu d'utiliser une norme L_2 , les auteurs de PVNet ont choisi d'utiliser la norme lissée ℓ_1 . Cette norme a une réponse constante pour de grandes différences comme la norme L_1 mais garde un profil quadratique entre 0 et 1. Il n'y a donc pas de discontinuité en 0 comme pour la norme L_1 .

$$\mathcal{L}_{\text{dir}}(\widehat{D}_{i,j}, D_{i,j}) = \sum_{k=0}^K \sum_{u=0}^W \sum_{v=0}^H \ell_1(\widehat{D}_{i,j,k,u,v} - D_{i,j,k,u,v}) \quad (7.73)$$

Conclusion

Dans cette section, nous avons détaillé les mesures et les fonctions de coûts utilisées pour évaluer et entraîner les réseaux de neurones de l'état de l'art. Pour la détection 3D, le principe dominant est de mesurer l'écart entre prédiction et étiquette. Cette mesure est réalisée par une fonction de distance, souvent Euclidienne (L_2), parfois simplifiée pour faciliter les calculs (SSD), ou adaptée pour les besoins de la descente de gradient (ℓ_1).

Nous avons aussi déjà présenté les jeux de données à notre disposition. Tout ce qu'il nous manque pour entraîner ces réseaux est un protocole de préparation des images. C'est précisément ce que nous allons présenter dans le chapitre suivant.

Chapitre 8

Préparation des images d'entraînements

Introduction

La préparation des images d'entraînement est une étape essentielle pour plusieurs raisons.

Premièrement, elle assure qu'aucun élément visible dans les images ne puissent biaiser l'entraînement, et donc biaiser nos expériences. Par exemple, des mires de calibrations de repères comme les marqueurs ChArUco sont utilisés dans le jeu de données LINEMOD. Ils sont utilisés pour calculer la position et l'orientation des objets dans les images d'entraînement. Les réseaux de neurones que nous allons entraîner risqueraient d'utiliser ces marqueurs pour apprendre, au lieu d'utiliser la partie visible des objets. Or, ces marqueurs ne seront pas présents en situation réelle. Le réseau de neurones pourrait donc ne pas généraliser ses résultats à cette nouvelle situation, i.e. à ces nouvelles images et serait, alors, incapable de fonctionner correctement. Nous devons donc effacer tous ces éléments avant de présenter les images au réseau de neurones pendant l'entraînement. Dans l'état de l'art des jeux de données, nous avons aussi mis en évidence certains objets hors-champs qu'il nous faut ignorer.

Deuxièmement, elle permet d'adapter les dimensions des images à la dimension attendue par le réseau de neurones utilisé. Les réseaux attendent généralement des images carrées de petites dimensions. Par exemple 224×224 pixels. Nous devons donc recadrer ou redimensionner les images avant de présenter les images au réseau de neurones pendant l'entraînement.

Dernièrement, il peut exister un écart entre les données d'entraînement et les données issues de situations réelles. Cet écart est appelé "*écart sémantique*". À cause de lui, le réseau de neurones peut se spécialiser sur les images d'entraînement, sur lesquelles il fonctionnera bien, très bien, même trop bien. Tellement, qu'il fonctionnera moins bien, ou pas du tout, sur les images issues de situations réelles. C'est ce qu'on appelle la "sur-spécialisation" ("*overfitting*" en Anglais). Pour palier à ce problème, la seule solution est d'assurer une bonne variété d'images d'entraînement. Nous devons donc augmenter autant que possible la quantité et la variété des images d'entraînement, soit en faisant l'acquisition de plus d'images, soit en modifiant les images à disposition. On peut, par exemple, recadrer de plusieurs façons les images, changer légèrement la luminosité, la netteté, voire changer complètement l'arrière-plan.

Dans ce chapitre, nous détaillons les prétraitements faits sur les images d'entraînement qui permettent de répondre à ces problèmes

8.1 Recadrage et échelle des images

Les réseaux de neurones convolutionnels utilisent des couches de réductions qui opèrent un sous-échantillonnage spatial des activations. Cette opération peut être réalisée grâce à



FIGURE 8.1 – Illustration de la substitution de l'arrière-plan d'une image. Des cadres aux dimensions des images sont choisis aléatoirement dans des images de substitutions et utilisés comme arrière-plan.

une couche dédiée comme par exemple le "*max pooling*". Ou bien, grâce au pas des couches de convolutions qui dont les applications du noyau sont espacés (au lieu d'être contiguës). La taille de l'entrée du réseau est souvent contrainte en fonction du nombre de réduction. Autrement les tenseurs seraient très grands et lourds en mémoire. De plus la première couche dense serait connectée à énormément d'entrées et donc elle comprendrait plus de paramètres. Pour limiter le nombre de paramètres du réseau et son empreinte mémoire la taille des images est généralement limitée. Par exemple, LeNet qui utilise 2 réductions n'accepte que des images de 32×32 , VGG qui utilise 5 réductions peut lui travailler avec des images de 224×224 pixels.

Pour contraindre une image à la taille d'entrée du réseau deux techniques sont envisageables :

1. On peut redimensionner l'image entièrement au détriment du ratio d'aspect original.
2. Ou on peut recadrer l'image ce qui élimine les bords.

Et on peut aussi combiner ces deux techniques. Nous précisons pour chaque entraînement réalisé quelle technique est utilisée et comment. Plus de détail sur le recadrage et la mise à l'échelle sont présentés aux sous-sections 2.1.6 et 2.1.7.

8.2 Substitution des arrière-plans

Comme mentionné en début de chapitre, les images des jeux de données pour la détection 3D figurent souvent des marqueurs de repères. Ces derniers doivent être effacés des images d'entraînement pour garantir que les réseaux que nous allons entraîner ne les utilisent pas. La meilleure solution est de remplacer les pixels où apparaissent les marqueurs par ce qui

se trouve derrière ou sous les marqueurs. Il s'agit peut-être d'un plateau, d'une table ou d'autre chose. Cette surface sur laquelle repose les objets peut d'ailleurs être visible ailleurs dans l'image, ou dans d'autres images. On peut donc envisager d'utiliser des techniques de remplissage automatique ("*inpainting*"). Si on connaît les pixels où apparaissent les marqueurs et que l'on dispose d'une photo de ce qui se trouve dessous ou derrière alors on peut les effacer directement par masquage. Malheureusement pour la plupart des jeux de données où sont présents des marqueurs, comme LINEMOD ou T-LESS, nous ne disposons pas des images d'arrière-plan.

La solution que nous avons choisie est de ne garder que l'avant-plan des objets comme c'est généralement fait dans l'état de l'art. Nous remplaçons tout le reste par une image choisie aléatoirement dans le jeu de données SUN [127],[128]. Les photos de SUN sont beaucoup plus grandes que les images recadrés que nous utilisons. Nous extrayons aléatoirement un cadre d'arrière-plan dans l'image d'arrière-plan ce qui permet d'augmenter la variété des arrière-plans et donc des images d'entraînement obtenues. Un exemple visuel est présenté à la figure 8.1.

La substitution de l'arrière-plan ne modifie ni les paramètres intrinsèques de la caméra, ni les paramètres extrinsèques des objets vus. Elle n'a donc aucun impact sur les étiquettes, quelles qu'elles soient. Elle a, néanmoins, un impact sur le réalisme des images car l'objet semble flotter dans les airs. De plus la qualité du masquage est directement impactée par la qualité du masque de segmentation. Dans les jeux de données pour l'estimation des six degrés de liberté, il est calculé grâce aux étiquettes de pose.

8.3 Ajustements Aléatoires

Nous utilisons un certain nombre d'augmentations qui ajuste les images d'entraînement. Ces augmentations sont calculées en ligne grâce à des fonctions fournies par TensorFlow. Elles agissent directement sur les valeurs des pixels pour simuler des variations de luminosité, de contraste, de teinte, de saturation, ou encore de dégradation d'image liées à leurs compressions au format JPEG.

8.3.1 Luminosité



FIGURE 8.2 – Illustration de l'ajustement de la luminosité sur une image. Avec un $\delta_L = -1.0$ l'image est complètement noire, et avec $\delta_L = +1.0$ elle est complètement blanche.

La luminosité est très importante dans les scénarios en extérieur. Dans ce cas, la principale source de lumière diffuse est le soleil. Cette lumière varie en intensité au fil de la journée, ou selon l'exposition au soleil.

Il est possible de simuler ces changements en jouant sur l'intensité des pixels d'une image. Pour ce faire, on convertit d'abord les images en valeurs flottantes dont les valeurs des pixels sont comprises entre 0 et 1 au lieu de 0 et 255. Puis, on choisit un nombre aléatoire compris lui aussi entre 0 et 1 que nous noterons δ_L . Ce nombre est ajouté à tous les canaux (sauf l'alpha) de tous les pixels de l'image. Enfin, on s'assure qu'aucun pixel n'ait une valeur inférieure à 0 ou supérieure à 1.

$$I_{i,u,v,c} = \text{clamp}(I_{i,u,v,c} + \delta_L, 0, 1) \forall c \in [r, g, b] \wedge \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (8.1)$$

La figure 8.2 illustre l'effet du paramètre δ_L sur une image.

8.3.2 Contraste

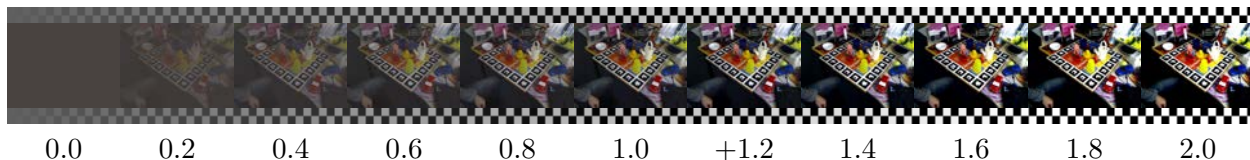


FIGURE 8.3 – Exemple de l'effet produit par l'ajustement du contraste sur une image. Avec un $\lambda_C = 0.0$ l'image est complètement unicolore, avec $\lambda_C = 1.0$ elle est identique à l'image d'origine. Avec $\lambda_C > 1.0$ la netteté de l'image les couleurs sont plus vive mais les objets unicolores perdent en détails car les couleurs de leurs pixels se retrouvent rapprochées. En effet l'augmentation global du contraste peut conduire à une diminution local de celle-ci. Par exemple, à un contraste élevé, on ne distingue plus les différentes faces de l'arrosoir, qui devient d'un blanc uni.

Le contraste d'une image quantifie l'écart entre le pixel le plus clair et le plus sombre d'une image, divisé par la somme des deux. Concrètement, une image avec peu de contraste aura des pixels dont les valeurs sont proches, alors qu'une image avec beaucoup de contraste aura des pixels avec des valeurs très éloignées. On discernera donc moins bien les objets dans une image avec moins de contraste, et mieux dans une image avec plus de contraste. Le contraste est lié aux objets dans l'image et à la façon dont ils sont éclairés. Par exemple, un échiquier aura un fort contraste propre à son apparence, mais s'il est peu éclairé, alors l'image observée aura peu de contraste.

Pour ajuster ce contraste aléatoirement lors de l'entraînement, on choisit un facteur aléatoire noté λ_C supérieur ou égale à 0. Puis, pour chaque canal de l'image, la nouvelle valeur des pixels est calculée comme suit :

$$I_{i,u,v,c} = (I_{i,u,v,c} - \mu(I_{i,c})) \times \lambda_C + \mu(I_{i,c}) \forall c \in [R, G, B] \wedge \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (8.2)$$

$$\mu(I_{i,c}) = \frac{1}{W \times H} \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} I_{i,u,v,c} \quad (8.3)$$

Enfin on s'assure qu'aucun pixel n'est une valeur inférieure à 0 ou supérieure à 1.

La figure 8.3 illustre l'effet du paramètre λ_C sur une image. Trop diminuer le contraste conduit à une image où tous les pixels ont la même couleur, égale à la couleur moyenne de l'image. Trop l'augmenter va réduire les contrastes locaux au profit du contraste global, ce qui peut entraîner la perte de certains détails sur les objets. Par exemple, à un contraste élevé, on ne distingue plus les différentes faces de l'arrosoir dans la figure 8.3.

8.3.3 Teinte

La teinte est la composante qui correspond à la couleur au sens longueur d'onde d'un pixel dans le modèle de couleur HSV ("*hue, saturation, value*" en Anglais). La couleur photographiée d'un objet dépend de la couleur de l'objet et de la lumière qui l'éclaire. La lumière du soleil, est par exemple, légèrement jaune. Celle de la lumière émise par des LED peut être



FIGURE 8.4 – Illustration de l'ajustement de la teinte sur une image. Avec un $\delta_C = 0.0$ l'image est identique. Toutes valeurs de δ_C supérieures ou inférieures à 0 va "décaler" la couleur de chaque pixel de l'image dans le spectre visible.

légèrement bleu. Les caméras de gamme moyenne, selon le format d'encodage, ne détectent que peu de couleurs. Entre deux images, la couleur d'un objet est susceptible de changer légèrement à l'image du fait de la discrétisation des couleurs et de la compression. Pour qu'une méthode de détection soit robuste, elle doit pouvoir s'accommoder de ces changements de couleurs. Étant donné que la teinte est une valeur comprise entre 0 et 1 et qu'elle modélise toutes les couleurs du spectre visible, le plus petit changement de teinte entraîne un important changement de la couleur dans l'image.

Pour ajuster la teinte, l'image est d'abord convertie au format *HSV* ; puis on ajoute au canal *H* de chaque pixel une valeur choisie aléatoirement, notée δ_T ; on s'assure encore qu'aucun pixel n'est une valeur inférieure à 0 ou supérieure à 1 ; et enfin l'image est reconvertie au format *RGB*.

$$I_{i,u,v,H} = \text{clamp}(I_{i,u,v,H} + \delta_T, 0, 1) \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (8.4)$$

La figure 8.4 illustre l'effet de ce paramètre sur une image du canard de LINEMOD. En particulier, on peut voir l'effet sur la couleur jaune du canard. Si $-0.01 < \delta_T < 0$ alors tous les objets jaunes seront "verdis" et si $0 < \delta_T < 0.01$ ils seront "rougis".

8.3.4 Saturation

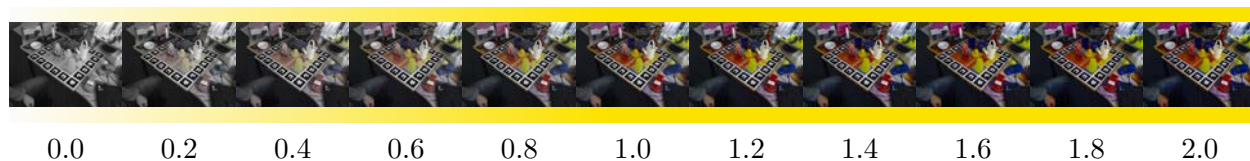


FIGURE 8.5 – Illustration de l'ajustement de la saturation sur une image.

La saturation est la deuxième composante du modèle de couleur *HSV*. Étant donné une couleur définie par sa teinte, la saturation est la vivacité ou la pureté de cette couleur. Concrètement, si on désature une image, elle sera complètement noire et blanche ($S = 0$). Au contraire, si on augmente la saturation, alors les couleurs seront plus vives.

Pour ajuster la saturation, on commence par convertir l'image au format *HSV* comme pour la teinte. On choisit aléatoirement la valeur d'un paramètre supérieure ou égale à 0 que l'on va multiplier à la saturation de l'image, nous le noterons λ_S . Comme précédemment, on s'assure qu'aucun pixel n'est une valeur inférieure à 0 ou supérieure à 1 et on reconvertit l'image au format *RGB*.

$$I_{i,u,v,S} = \text{clamp}(I_{i,u,v,S} \times \lambda_S, 0, 1) \forall u \in [0, W[\wedge \forall v \in [0, H[\quad (8.5)$$

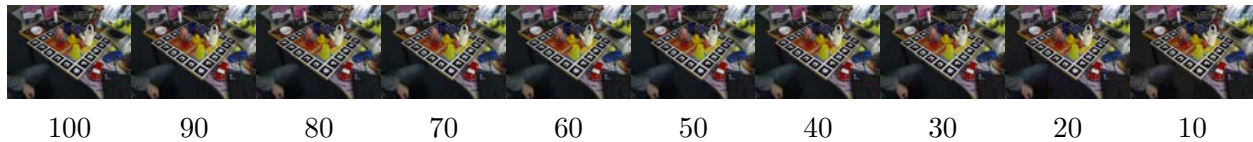


FIGURE 8.6 – Illustration de l'application de la compression JPEG à différents niveaux de qualités sur une image.

8.3.5 Qualité JPEG

La qualité JPEG est différente des autres augmentations, car elle n'est pas liée à l'objet ni à la lumière qui l'éclaire. Elle est utilisée pour répondre à un biais logiciel qui est introduit dans les images. En effet, les caméras, sauf si elles filment dans un format brute (format *RAW*), compressent les images pour les stocker de façon plus compacte. La compression est aussi utilisée pour transmettre les images sur un réseau plus rapidement. JPEG est un format d'image utilisé par les caméras et les logiciels d'édition d'image pour enregistrer les images et gagner de l'espace disque en les compressant. C'est aussi le format le plus utilisé sur internet. L'algorithme de compression JPEG découpe l'image en blocs de 8×8 pixels, puis calcule la transformée discrète en cosinus de chaque bloc. Vient ensuite l'étape de quantification où les hautes fréquences vont être filtrées pour gagner de la place. Plus le niveau de compression est élevé, plus on filtre de fréquences dans les blocs. Mais plus on a éliminé les hautes fréquences, plus d'informations et de détails sont perdus, et plus on verra des artefacts résiduels apparaître dans l'image décodée. En effet, si on ne garde qu'une fréquence par bloc, cela revient à considérer que le bloc est mono-couleur.

Si une méthode de détection est appliquée dans un logiciel ou une application dans laquelle la compression JPEG est utilisée, alors, il est important de préparer la méthode aux artefacts laissés par la compression JPEG. Pour ce faire, nous compressons l'image avec une qualité dont la valeur est choisie aléatoirement. Nous noterons cette valeur τ_{JPEG} . L'image est ensuite décompressée et des artefacts de compression apparaîtront. La figure 8.6 présente les effets de la compression JPEG à différents niveaux de qualité.

8.4 Conclusion

Grâce à ces étapes de préparation des images nous assurons une bonne variété d'images lors de l'entraînement des réseaux de neurones que nous considérons. Les réseaux pourront être robustes aux changements de luminosité, de contraste, de teinte, de saturation et même aux effets de la compression JPEG. La substitution des arrière-plans est appliquée dans tous les jeux de données pour garantir qu'aucune mire de calibration ne biaise l'entraînement. De plus, elle permet d'augmenter le nombre d'images d'entraînement.

Chapitre 9

Résultats de détections

9.1 Détection 1D

Introduction

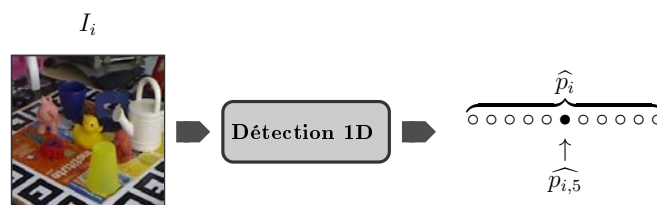


FIGURE 9.1 – Schéma illustrant un module de détection 1D. Les probabilités que chaque catégorie d’objets soit présente à l’image sont estimées. Ce qui permet de savoir quels objets sont présents dans l’image.

Dans le but de mettre en œuvre des applications de réalités mixtes à destination d’utilisateurs industriels nous avons besoin d’un système de vision par ordinateur capable de reconnaître et d’estimer les positions d’objets présents à l’image. Il est possible de résoudre ces deux problèmes simultanément, mais dans la plupart des cas les objets sont premièrement reconnus, puis leurs positions sont calculées dans un deuxième temps. La première étape est donc de détecter quels objets sont présents dans l’image, il s’agit d’un problème de classification d’images. Ce module est illustré en figure 9.1.

L’état de l’art nous a montré que la technique la plus probante pour mettre en place un tel système de classification d’images consiste à employer un réseau de neurones. Cette méthode consiste à entraîner de façon supervisée le réseau grâce à des images et leurs annotations.

Puisque notre objectif final est la détection 3D, nous avons utilisé des jeux de données d’estimation des six degrés de liberté qui peuvent aussi être utilisés pour la classification d’images. Notamment LINEMOD, T-LESS et YCB-VIDÉO. Ces jeux de données de détection 3D présentent généralement moins de facteurs de difficultés que les jeux de données de détection 1D et 2D classiques (ILSVRC, COCO). Par exemple, ils comprennent moins de catégories d’objets. LINEMOD en propose 11, T-LESS 30, ce qui est peu comparé aux 1000 de ImageNet [25] (21841 si on considère les sous catégories d’objets) ou aux 80 de COCO [74]. De plus, dans un jeu de données de détection 3D, les photographies sont prises avec une seule caméra et non sourcées sur internet, la perspective est donc la même dans toutes les images. Ils sont généralement photographiés en laboratoire, les conditions d’éclairage sont donc contrôlées et favorables et l’environnement est moins varié. Les réseaux que nous allons entraîner auront donc à faire face à moins de critères de difficultés.

De ce fait, nous n'avons pas entraîné les réseaux au sommet de l'état de l'art comme ViT [27], ou les versions les plus larges de EfficientNet [113]. Certains de ces réseaux ne peuvent être entraînés qu'au prix d'énormément de puissance de calculs, demandant plusieurs serveurs équipés de plusieurs GPUs. Nous souhaiterions déployer notre application sur des casques de réalité mixte dont la mémoire et les capacités de calculs sont limitées. Nous avons donc privilégié des réseaux de classification d'images utilisant moins de ressources. Notre choix s'est aussi porté sur des réseaux servant d'ossature à des réseaux de détections 2D et 3D. Nous avons retenu en particulier AlexNet, VGG-19, et GoogLeNet dont les résultats sont présentés aux sous-sections suivantes.

9.1.1 VGG-19

Nous avons commencé nos expériences avec VGG qui est le réseau incontournable en vision par ordinateur. Son architecture a été utilisée pour résoudre des tâches comme la segmentation sémantique, la détection d'objets, l'estimation de la pose et bien sûr la classification d'images pour laquelle il a été proposé en premier lieu. Nous avons utilisé la configuration "E" qui comprend 19 couches dont 16 couches de convolution et 3 couches denses.

LINEMOD

Pour commencer, nous avons appliqué VGG sur LINEMOD qui est le jeu de données le plus utilisé en détection 3D. De par sa profondeur et son nombre de paramètres, VGG ne devrait avoir aucun mal à identifier les 11 objets de LINEMOD.

Les images de LINEMOD sont divisées en deux groupes : 75 % pour l'entraînement et 25 % pour la validation et les tests. Les proportions d'images d'entraînements et de validations sont respectées au sein des catégories d'objets pour éviter de déséquilibrer l'entraînement à la faveur de catégories plus représentées que d'autres. Les images sont recadrées à 224×224 pixels et centrées sur l'objet étiqueté. L'arrière-plan de chaque image est remplacé par une image du jeu de données SUN. Au total, il y a 60048 images d'entraînement et 20034 de validation. Le réseau est entraîné pendant 1000 époques.

VGG obtient de très bons résultats sur LINEMOD avec une précision de 99.92 % et un rappel de 99.88 %. Les détails de cet entraînement sont présentés en annexe A.1.

T-LESS

Les objets de T-LESS, bien que peu nombreux, se sont montrés bien plus difficiles à classifier que ceux de LINEMOD. Des similitudes existent entre plusieurs objets, et selon le point de vue, ils peuvent être impossibles à différencier car ils partagent certaines pièces. La plupart des objets ont aussi peu de texture. Pour ajouter à ces difficultés, les images d'entraînement sont photographiées dans un environnement très contrôlé alors que celles de tests proposent des situations complexes.

Entraînement initiale Comme pour LINEMOD, nous avons d'abord essayé un protocole très simple. Nous avons recadré les images 384×384 pixels, puis nous les avons redimensionnées à 224×224 pixels, et pour finir, nous effaçons l'arrière-plan avec une image de SUN. Au total, il y a 37584 images d'entraînement et 69552 de test.

Cet entraînement nous a montré que VGG est capable de reconnaître les objets TLESS, car sa précision sur les images d'entraînement est de 99.46 %. Malheureusement, du fait de l'écart sémantique entre images d'entraînements et de tests, VGG ne généralise pas bien. Sur les images de test, la précision ne dépasse pas les 28.58 % et le rappel les 16.22 %. VGG

produit en particulier beaucoup de faux négatifs (75.54 %). Il faut donc mettre en place des augmentations d'images pour combler cet écart.

Limite d'occlusion T-LESS propose des images avec des objets fortement, voir complètement occultés. Pour ce deuxième entraînement, nous avons donc éliminé les exemples de test où les objets sont occultés à 70 % ou plus. Ce qui produit 62681 images de test, soit 6871 images de test ignorées (9.88 %). La préparation des images et l'optimisation des poids du réseau sont réalisées de façon identique au premier entraînement. Cet entraînement donne des résultats très similaires. La précision du réseau est légèrement inférieure : 25.45 % contre 28.58 % précédemment. Mais le rappel a progressé : 0.1758 contre 0.1622. Cette progression est due au fait que les objets complètement occultés introduisaient des faux négatifs. La mesure qui leur est consacrée a progressé, passant de 24.46 % à 24.76 % ce qui veut dire qu'il y a moins de faux négatifs.

Recadrage et échelle Les objets peuvent se trouver plus loin dans les images de tests. Pour le troisième entraînement, nous avons essayé de varier aléatoirement la taille de recadrage. Nous partons toujours d'une taille de 384×384 , que nous multiplions par un facteur aléatoire de plus ou moins 5 %, i.e. choisi entre 95 % et 105 %. Cette façon de préparer les images d'entraînement donne de bien meilleurs résultats avec une progression de la précision de 5 % et du rappel de 1 % qui s'établissent à 30.16 % et 18.57 % respectivement.

Dans le cadre d'une expérience de classification d'images, la méthode de détection n'est pas censée connaître, ni utiliser, le rectangle encadrant des objets. Même chose pour les images de test lors d'une expérience de localisation d'objets ou d'estimation des six degrés de liberté d'un objet. Mais, si tout ce que l'on souhaite est d'affiner les poids d'un réseau, alors, il peut être justifiable de les utiliser. En recadrant les images aux dimensions du rectangle encadrant, plus une marge de 16 pixels dans toutes les directions, nous mettons l'objet à classifier au centre de l'image et nous lui faisons occuper au plus l'image recadrée. Cette façon de recadrer les images permet de faire progresser la précision sur les images de tests de 16 %. Le rappel est aussi très nettement meilleurs à 34.94 %, contre 18.57 % précédemment.

Si, en plus, on varie aléatoirement la taille de recadrage par un facteur choisie entre 0.95 et 1.05 comme précédemment, alors la précision dépasse enfin les 53.30 %. Le rappel est lui légèrement meilleur à 36.82 %.

Condition d'éclairage Les images d'entraînement sont photographiées dans un environnement très contrôlé, ce qui n'est pas le cas des images de test. De ce fait, les images de test ont un éclairage différent des images d'entraînement. Pour compenser ce changement, nous avons essayé d'appliquer plusieurs augmentations sur les images d'entraînement. Nous modifions la luminosité, le contraste, la teinte et la saturation aléatoirement. Comparé au deuxième entraînement, les résultats sont meilleurs. La précision a nettement augmenté, passant de 25.45 % à 29.98 % et le rappel a augmenté de façon marginale.

Tout ensemble Pour ce dernier entraînement, nous appliquons les préparations d'images ayant donné les meilleurs résultats ci-dessus. Nous filtrons les exemples occultés à plus de 30 %. Nous recadrons les images aux dimensions des rectangles encadrants, plus une marge de 16 *px*, fois un coefficient choisi aléatoirement entre 0.9 et 1.1. Nous effaçons les arrière-plans avec des images de SUN. Et enfin, nous appliquons des changements aléatoires de luminosités, de contrastes, de teintes et de saturations. À la fin de cet entraînement la précision est moins bonne : 51.12 % contre 53.30 %. Mais le rappel est meilleur 37.91 % contre 36.82 %.















		VGG-19 / T-LESS						
		#1	#2	#6	#3	#4	#5	#7
								
map		21.85 %	21.80 %	23.68 %	24.52 %	40.36 %	42.54 %	42.30 %
p		28.61 %	25.31 %	29.98 %	30.16 %	46.50 %	53.30 %	51.12 %
r		16.29 %	17.49 %	17.65 %	18.57 %	34.94 %	36.82 %	37.91 %

TABLE 9.1 – Comparatif des résultats obtenus avec plusieurs méthodes de préparation des images et VGG-19 sur T-LESS. Légende :  Filtrage des exemples dont la visibilité est inférieure à 30%.  Soustraction de l’arrière-plan et des distracteurs.  Recadrage manuel à une taille fixe de 384×384 pixels.  Recadrage manuel à une taille fixe de 384×384 pixels, fois un facteur aléatoire.  Recadrage à la taille du rectangle encadrant plus une marge de 16 pixels.  Recadrage à la taille du rectangle encadrant plus une marge de 16 pixels et fois un facteur aléatoire.  Variations aléatoires de luminosités, contrastes, teintes et saturations.

Conclusion Au cours de ces entraînements, le coût sur les images d’entraînement a toujours été bien minimisé, la précision et le rappel ont tous deux atteint de bonnes valeurs proches de 1. Ce qui démontre que VGG est tout à fait capable de reconnaître les objets de T-LESS. Mais VGG généralise difficilement sur les images de tests de T-LESS.

La différence de protocole dans l’acquisition des images de test a introduit un écart sémantique important entre images d’entraînement et de test. Les augmentations d’images permettent quelques améliorations sur les valeurs des mesures de performances. Nous avons ainsi réussi à faire progresser la précision de près de 24.69 % et le rappel de 20.53 %. Le tableau 9.1 détaille les gains réalisés avec chaque entraînement. Filtrer les exemples trop occultés a permis de faire progresser le rappel de 1 %. Varier aléatoirement la taille du rectangle de recadrage a boosté la précision de 5 %. Recadrer à la taille des rectangles encadrants s’est avéré essentiel puisque la précision a bondi de 16.34 % sans variation aléatoire de tailles, et 23.14 % avec. Varier aléatoirement la luminosité, le contraste, la teinte et la saturation permet de faire progresser encore le rappel à 37.91 % au prix de 2 % de précision.

Mais on observe toujours une divergence inévitablement entre images d’entraînement et de test, ce qui montre que VGG se spécialise sur les images d’entraînements et ne généralise pas sur les images de tests. Les similarités entre certains objets ne sont tout simplement pas prises en compte dans les étiquettes. De ce fait, le réseau commet de nombreux faux positifs et faux négatifs. Une solution possible serait d’utiliser WordTree comme représentation [80, 92] pour les étiquettes.

Tous les détails des entraînements réalisés sont à retrouver aux annexes A.2 à A.8.

YCB VIDÉO

YCB VIDEO propose 21 objets. Encore une fois, c’est peu comparé à ILSVRC ou COCO, mais le jeu de données propose d’autres facteurs de difficultés. Certains objets sont peu texturés (mug, bol, pichet) et d’autres sont identiques à un facteur d’échelle près comme les deux pinces. De plus, certaines images sont extraites d’un flux vidéo au lieu d’être photographiés individuellement. Les images comportent donc des facteurs susceptibles de perturber l’entraînement comme le flou de mouvement ou l’effet d’obturation. Lors de la préparation des images, nous excluons les objets occultés à plus de 30 %. Au total, il y a donc 515 350 objets uniques pour l’entraînement et 97 242 pour les tests.

Premier entraînement Pour le premier entraînement, nous recadrons les images centrées sur l’objet à 384×384 puis nous les redimensionnons à 224×224 pixels. Lors de l’ap-










VGG-19 / YCB-VIDÉO			
	#1	#2	#3
	  	  	  
p	38.17 %	77.38 %	94.06 %
r	12.21 %	69.49 %	90.92 %

TABLE 9.2 – Comparaison des résultats obtenus par VGG sur YCB VIDÉO selon la méthode de préparation des images.






VGG-19 / YCB-VIDÉO		
	#3	#3
	  	 
map top 1	92.3490 %	85.2307 %
map top 2		92.4415 %
map top 5		97.8209 %

TABLE 9.3 – Pourcentage d’images de YCB-VIDÉO correctement classifiées par VGG. Dans la première colonne les arrières plan et les distracteurs sont effacé. Dans les colonnes suivantes les arrières plan et les distracteurs ne sont pas effacés.

prentissage, les images sont augmentées en variant la teinte, la saturation, le contraste et la luminosité. Nous ne remplaçons pas les arrière-plans, car aucun marqueur n’est visible dans les images de ce jeu de données. VGG apprend correctement à classifier les images d’entraînement, mais pas celles de test. Il commet beaucoup de faux négatifs ($\sim 80\%$) mais peu de faux positifs $\sim 1\%$. Si on observe les résultats obtenus sur les images, on remarque que lorsque les images sont recadrées à 384×384 , beaucoup d’objets sont inclus en plus de celui au centre. Lors de l’entraînement, on demande donc au réseau d’associer plusieurs objets visibles à une seule étiquette. De plus, les images de test sont tirées de séquences vidéo inédites où la disposition des objets est inconnue pour le réseau.

Pour résoudre ce problème, dans le cas d’une expérience de classification d’images, une solution est de modifier les étiquettes lors du recadrage des images. Si plusieurs objets sont présents dans le cadre, ils doivent être indiqués comme tels.

Deuxième et troisième entraînements Pour vérifier que c’est bien ce problème qui est en cause, nous avons réalisé deux entraînements supplémentaires. Pour le deuxième, nous recadrons les images aux dimensions du rectangle encadrant ce qui devrait limiter la présence d’autres objets, sauf dans le cas des occlusions en avant plan. Dans ces conditions, la précision atteint les 77 % et le rappel les 69 %. Comme pour le premier entraînement, on constate qu’il y a beaucoup de faux négatifs ($\sim 25\%$) mais peu de faux positifs ($\sim 1\%$). Pour le troisième et dernier entraînement, nous avons en plus effacé l’arrière-plan et les autres objets avec des images de SUN. Dans ces conditions, la précision dépasse les 94 % et le rappel les 90 %. Ce sont donc bien les autres objets dit "distracteurs" qui impactent négativement le premier et deuxième entraînement.

Approfondissement des résultats Pour approfondir ces résultats, nous avons utilisé VGG après le troisième entraînement et calculé le nombre moyen d’images correctement classifiées (MAP) avec plusieurs façons de préparer les images. Avec effacement des arrière-plans et des distracteurs, la MAP top-1, où seul l’objet avec la plus forte probabilité est considéré comme positif, est de 92,3 %. Si on désactive l’effacement des arrière-plans, la MAP chute de $\sim 7\%$. Cette baisse de performance est due principalement aux occlusions avec des

distracteurs, devant ou derrière l'objet étiqueté. La précision de VGG est aussi impactée par des objets très similaires comme les deux pinces. Nous avons donc aussi mesuré la MAP top-5 (sans remplacer les arrière-plans) qui est de 97.8, soit un résultat presque parfait. Mais autoriser des confusions de l'ordre de 5 objets est très "large" quand on n'a que 21 objets à différencier. Nous avons donc calculé la MAP top 2 spécialement pour le cas des pinces. Elle atteint 92.4%, ce qui est meilleur que les 92.3% initiaux et surtout bien meilleur que les 77% du deuxième entraînement et des 38% du premier entraînement. Les résultats sont présentés au tableau 9.3.

Conclusion VGG est capable de reconnaître les objets de YCB-VIDÉO comme le montre les très bons résultats obtenus. Il est néanmoins important de refléter la présence de tous les objets dans les images, surtout dans celles d'entraînement au risque d'impacter négativement les résultats. À défaut, il faut effacer les distracteurs pour pouvoir correctement entraîner VGG.

Conclusion sur VGG VGG est à même de reconnaître les objets de tous les jeux de données que nous lui avons soumis. Ces entraînements nous ont montré qu'il est très important pour TLESS et YCB VIDÉO d'avoir un processus de préparation des images cohérent. Ce processus doit associer des étiquettes qui décrivent avec exactitude les objets présents dans les images. Il doit aussi augmenter suffisamment les images pour adresser tout vide sémantique présent.

9.1.2 AlexNet

AlexNet est le premier réseau à avoir obtenu un bon score de classification sur ILSVRC 2012. C'est un petit réseau de neurones convolutionnel comportant 5 couches de convolution et 3 couches denses. Comparé à VGG, il est bien moins adapté pour les 1000 objets et la variété des images d'ImageNet. Mais pour des jeux de données avec moins d'objets et moins de facteurs de difficultés, il peut être tout à fait pertinent. C'est pourquoi nous avons évalué ses performances sur les jeux de données que nous avons retenus.

LINEMOD

AlexNet est entraîné sur les images de LINEMOD tel que nous les avons préparées pour VGG. 75% sont utilisées pour l'entraînement et le reste pour la validation et les tests. Elles sont recadrées à 224×224 pixels puis les arrière-plans sont remplacés par des images de SUN. Le réseau est entraîné pendant 1000 époques avec l'algorithme de descente de gradient stochastique. AlexNet se prête très bien à la classification des 11 objets de LINEMOD. La précision et le rappel croissent rapidement et dépassent les 99% à la fin de l'entraînement.

T-LESS

Pour entraîner AlexNet sur le jeu de données T-LESS, nous avons utilisé la technique de préparation des images qui avait donné les meilleurs résultats avec VGG. Premièrement, nous filtrons les objets occultés à plus de 30%. Nous recadrons ensuite les images à la taille du rectangle encadrant, plus une marge de 16 pixels, fois un facteur aléatoire compris entre 0.9 et 1.1. Nous effaçons l'arrière-plan avec des images de SUN. Et enfin, nous appliquons des variations aléatoires de luminosité, contraste, teinte et saturation. AlexNet est entraîné pendant 100 époques à l'aide de l'algorithme de descente de gradient stochastique. Nous utilisons un taux d'apprentissage fixe de 0.001. AlexNet généralise bien sur les images de

test avec une précision de 71.63 % et un rappel de 47.72 %. Encore une fois, le réseau produit beaucoup de faux négatifs, mais peu de faux positifs comme le montre les mesures qui leur sont consacrées : 57.58 % et 99.34 % respectivement. Comme VGG, AlexNet a peu confiance pour certaines de ses prédictions, mais ses résultats sont meilleurs que ceux de VGG. La précision d'AlexNet sur T-LESS est supérieure à celle de VGG qui est de 51.12 %.

YCB VIDÉO

Pour à nouveau pouvoir comparer AlexNet à VGG, nous préparons les images de TCB-VIDÉO de la même manière que précédemment. Les objets occultés à plus de 30 % sont ignorés. Les images sont recadrées aux dimensions des rectangles encadrants, plus une marge de 16 pixels et fois un coefficient aléatoire compris entre 0.9 et 1.1. Puis les arrière-plans sont effacés à l'aide d'image de SUN. Et enfin, nous appliquons des variations aléatoires de luminosité, contraste, teinte et saturation. La descente de gradient stochastique avec un taux d'apprentissage de 0.001 est utilisée pour optimiser les poids d'AlexNet. Au total, nous réalisons 20 époques d'entraînement, comme pour VGG, ce qui est suffisant vu le nombre d'images d'entraînement. Le réseau généralise encore une fois bien sur les images de test comme le montre les résultats à la fin de l'entraînement. La précision atteint 94.09 % et le rappel 90.94 % ce qui est très légèrement meilleur que VGG.

Conclusion sur AlexNet

AlexNet comprend 62 378 344 paramètres alors que VGG-19 en comprend 143 667 240, soit 2.3 fois moins. Néanmoins, vu que les jeux de données considérés comprennent moins d'objets comparés à ILSVRC, ses performances sont tout à fait acceptables. Sur les images d'entraînement de LINEMOD, AlexNet est légèrement moins bon. Il obtient une précision 99.16 % alors que VGG obtient 99.46 %. Même chose pour T-LESS où AlexNet obtient 99.16 % et VGG 99.39 %. Sur celle de YCB-VIDEO, la précision est identique à celle de VGG : 0.9992, mais le rappel d'AlexNet est légèrement moins bon : 99.92 % contre 99.66 %. AlexNet est donc suffisant pour modéliser les objets de ces trois jeux de données, même si VGG-19 reste meilleur.

Ce résultat se confirme sur les images de test de LINEMOD où AlexNet obtient une précision de 99.86 % alors que VGG-19 obtient 99.92 %. Sur YCB-VIDÉO, AlexNet est très légèrement meilleur que VGG avec une précision de 94.09 % contre 94.06 % pour VGG. Sur T-LESS l'écart est plus marqué. À même méthode de préparation des images d'entraînement, AlexNet obtient une précision de 71.63 % alors que VGG-19 obtient 42.30 %. Il est difficile d'expliquer une telle marge. Une explication possible est que AlexNet, du fait de son moins grand nombre de paramètres, se spécialise moins sur les images d'entraînement. Les objets de T-LESS étant relativement simples, il généraliserait alors mieux sur les images de tests. Il apprend peut-être des filtres plus génériques.

9.1.3 GoogLeNet

GoogLeNet est un réseau dit mobile. Grâce à des convolutions avec des noyaux de taille 1×1 le nombre de canaux des tenseurs sont compressés avant les convolutions 3×3 . Cela permet de réduire le nombre de connexions dans les couches 3×3 et donc de réduire le nombre de paramètres du réseau. L'empreinte mémoire du réseau est donc optimisée. Puisqu'il utilise moins de mémoire, GoogLeNet peut être implémenté sur des appareils mobiles comme des smartphones, tablettes, ou sur des ordinateurs mono-cartes utilisés en robotique. La mémoire

est aussi limitée sur les casques de réalités mixtes, GoogLeNet est donc intéressant dans notre cas.

LINEMOD

Pour le jeu de données LINEMOD, nous avons entraîné GoogLeNet dans les mêmes conditions que AlexNet et VGG. En fin d'entraînement, GoogLeNet obtient des scores tout aussi quasi parfaits avec une précision de 99.95 %. Le réseau a commis 14 erreurs seulement au total. Quelques erreurs sont dues à des objets distrayeurs qui viennent occulter l'objet d'intérêt. Dans une image, les oreilles du chat passent devant le téléphone, mais le masque de segmentation ne tient pas compte de ce détail, ce qui induit GoogLeNet en erreur. D'autres erreurs entre le téléphone, la lampe et la colle sont dues au fait que ces trois objets sont majoritairement blancs et sans texture ce qui pousse le réseau à les confondre.

T-LESS

Pour pouvoir à nouveau comparer GoogLeNet à AlexNet et VGG nous l'avons entraîné sur T-LESS, en préparant les images comme précédemment. Premièrement nous filtrons les objets occultés à plus de 30 %. Nous recadrons ensuite les images à la taille du rectangle encadrant, plus une marge de 16 pixels, fois un facteur aléatoire compris entre 0.9 et 1.1. Nous effaçons l'arrière-plan avec des images de SUN. Et enfin, nous appliquons des variations aléatoires de luminosité, contraste, teinte et saturation.

Cette fois-ci, les résultats sur les images de tests sont bien moins bons. La précision chute à 28.36 % alors qu'AlexNet obtient 71.63 % et VGG 51.12 %. GoogLeNet s'est fortement spécialisé sur les images d'entraînement et il ne généralise pas à celles de test.

YCB-VIDEO

Encore une fois, nous avons préparé les images de YCB-VIDÉO comme pour AlexNet et VGG. Les objets occultés à plus de 30 % sont ignorés. Les images sont recadrées aux dimensions des rectangles encadrants, plus une marge de 16 pixels et fois un coefficient aléatoire compris entre 0.9 et 1.1. Puis les arrière-plans sont effacés à l'aide d'image de SUN. Et enfin, nous appliquons des variations aléatoires de luminosité, contraste, teinte et saturation. Alors qu'AlexNet obtient une précision de 94.09 % sur ces images, et VGG-19 94.06 %, GoogLeNet obtient lui 94.45 %, soit le meilleur des 3 avec une marge bien plus significative qu'entre les deux précédents.

Conclusion sur GooLeNet

GoogLeNet est tout à fait à même de reconnaître les objets de LINEMOD, T-LESS et YCB-VIDEO. Sur LINEMOD et YCB-VIDEO, le réseau donne de meilleures performances comparé à VGG-19 et AlexNet. Ses performances sont notamment bluffantes sur LINEMOD où il ne commet que quatorze erreurs top-1. Par contre, sur T-LESS, le réseau donne les moins bonnes performances des trois.

Ces bons résultats sont dus au fait que GoogLeNet est un réseau extrêmement bien optimisé. Grâce aux modules "inception v2" et aux sorties auxiliaires, la convergence du réseau est améliorée. Cela se constate avec le coût qui décroît plus rapidement sur les images d'entraînement que pour AlexNet ou VGG. Mais s'il existe un vide sémantique important entre images d'entraînement et test, alors cet avantage se retourne contre lui, car les filtres appris seront plus spécialisés.

Conclusion

Dans cette section nous avons présenté les résultats de détection 1D de VGG-19, d’AlexNet et de GoogLeNet sur les jeux de données LINEMOD, T-LESS et YCB-VIDÉO. Le tableau 9.4 ci-contre regroupe ces résultats.

Sur ces jeux de données LINEMOD et YCB-VIDEO proposant une dizaine d’objets, GoogLeNet donne les meilleurs résultats. Cela est rendu possible grâce à une utilisation judicieuse des couches de convolution, qui permet d’optimiser le nombre de paramètres du réseau. VGG-19 et AlexNet donnent aussi de très bons résultats sur ces deux jeux de données, mais utilisent plus de paramètres.

T-LESS est le jeu de données le plus compliqué à adresser. Les trois réseaux donnent de moins bons résultats sur ce dernier et ce même avec une forte augmentation des images d’entraînement. D’après nous, deux propriétés de T-LESS amènent à ces résultats. Premièrement, il y a des différences très marquées entre images d’entraînement et de test comme nous l’avons expliqué au chapitre consacré à l’état de l’art de ce jeu de données. Cet écart sémantique requiert d’appliquer des augmentations d’images pour réduire l’écart, dont les paramètres doivent être choisis avec soin, comme les facteurs d’échelles et de teintes aléatoires. Deuxièmement, certains des objets de T-LESS sont très similaires et certains sont même composés de pièces identiques. Or, les étiquettes pour la classification d’images ne reflètent pas cette caractéristique des objets.

9.2 Détection 2D

Introduction

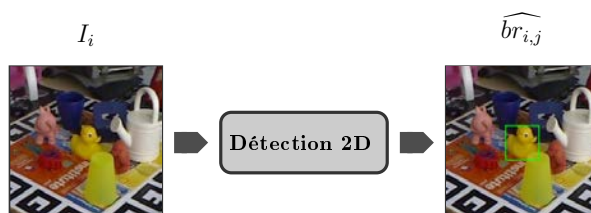


FIGURE 9.2 – Schéma illustrant un module de détection 2D. Les positions des objets à l’image sont estimés par des rectangles encadrants.

La deuxième tâche que nous avons considérée est la détection 2D. Grâce à la détection 2D, aussi appelée détection d’objets ou localisation d’objets, la position des objets dans une image est calculée. Cette information est suffisante pour mettre en place des applications de réalité augmentée. Par exemple, avec la détection 2D d’un objet, on va pouvoir afficher des informations à côté de cet objet dans l’image. La localisation d’objet est aussi très utilisée en vidéo-surveillance. Néanmoins, cette information ne permet pas d’incruster des informations en 3D dans l’environnement de l’utilisateur.

	VGG-19	AlexNet	GoogLeNet
LINEMOD	99.92%/99.88%	99.85%/99.78%	99.95%/99.91%
T-LESS	51.12%/37.91%	71.63%/47.72%	28.36%/15.17%
YCB-VIDÉO	94.06%/90.92%	94.09%/90.94%	94.45%/93.17%

TABLE 9.4 – Résultats de classifications d’images de VGG-19, AlexNet et GoogLeNet sur les jeux de données LINEMOD, T-LESS et YCB-VIDÉO.

L'état de l'art nous a montré que Mask R-CNN est le réseau qui offre les meilleures performances. C'est un énorme réseau qui répond aussi au problème de la segmentation sémantique. Il n'est pas possible de déployer Mask R-CNN sur des appareils mobiles ou des casques. Même implémenté sur un PC fixe Mask R-CNN ne donne pas de résultats en temps réel (5 ips dans l'article de 2018). Notre choix s'est donc porté sur un réseau donnant des résultats temp réel : YOLO.

De plus, la détection 2D n'est pas notre objectif final. Nous souhaitons à terme mettre en œuvre des réseaux de détection 3D. YOLO a déjà été utilisé comme ossature pour SSD6D dans ce but. Implémenter YOLO nous simplifierait donc le développement de SSD6D par la suite.

9.2.1 YOLO

YOLO fut le premier réseau à apporter une solution temps réel au problème de la détection d'objet. Ses performances sont atteintes grâce à une formulation inédite des données cible qui découpe l'image en plusieurs cellules, et une architecture adaptée. La deuxième version de ce réseau est tout aussi bonne que Faster R-CNN, mais nécessite beaucoup moins de ressources. Ce réseau est donc particulièrement intéressant dans le cadre d'une application sur casque de réalité mixte. De plus, la version 2 de YOLO a inspiré une solution au problème de la détection 3D nommé SSD6D comme nous l'avons vue dans l'état de l'art.

On rappelle que YOLO divise l'image en cellules. Chaque cellule prédit la catégorie de l'objet (sous la forme d'un vecteur logique) et deux rectangles. Les positions des deux rectangles sont définies relativement dans la cellule et leurs tailles sont prédites par rapport à la taille de l'image. Une probabilité appelée "objectness" est aussi associée à chaque rectangle et modélise la confiance du réseau dans sa prédiction. Lors de l'entraînement, cette valeur est comparée avec l'IoU entre étiquette et prédiction.

LINEMOD

La détection d'objets demande de préparer les images différemment de la classification d'images. Puisqu'on souhaite localiser les objets dans l'image, nous avons décidé de ne pas recadrer les images séparément pour chaque objet. Nous remplaçons seulement l'arrière-plan de toute l'image par une image de SUN pour effacer les autres objets non étiquetés et les marqueurs de pose. L'architecture de YOLO intègre une couche chargée de redimensionner les images à la taille requise pour les convolutions suivantes. Il n'est donc pas nécessaire de redimensionner les images lors de leurs préparations.

LINEMOD présente 15 objets différents ($C = 15$) mais un seul est visible et étiqueté par image. Nous n'utilisons donc qu'une seule cellule ($S = 1$) avec deux rectangles prédits comme dans l'implémentation des auteurs ($B = 2$). YOLO prédit donc $1 \times 1 \times (15 + 2 * (4 + 1))$ valeurs.

Nous entraînons YOLO sur 1000 époques en utilisant la descente de gradient stochastique et un taux d'apprentissage de 0.001. Tous les détails de l'entraînement sont à retrouver en annexe A.18. Les différents termes de la fonction de coût décroissent correctement durant l'entraînement. À la fin de l'entraînement, tous les termes sont extrêmement bas sur les images d'entraînement et encore plus bas sur les images de validations. YOLO généralise donc très bien sur les nouvelles images et ne se sur-spécialise pas. Le réseau est bon pour localiser les objets, mais commet des erreurs de classification. C'est cohérent à la remarque faite par les auteurs dans leur article : la distance des carrés est une bonne expression de l'erreur pour la position et la taille des rectangles, mais pas pour le vecteur logique utilisé pour la classification des objets. Dans la deuxième version de YOLO, ils ont d'ailleurs

utilisé la cross-entropie catégorique. On peut aussi remarquer que les deux détecteurs de la cellule YOLO se sont spécialisés sur le même ratio d'aspect. Cela pénalise les résultats de localisation. Par exemple le fer à repasser est correctement identifié, mais son rectangle n'est pas bien dimensionné. Dans LINEMOD, la plupart des objets sont plus hauts que larges et ces objets ont dominé l'entraînement. Pour résoudre ce problème, les auteurs ont introduit à partir de la deuxième version des ancres, qui sont des a priori de tailles. Elles contraignent et guident l'entraînement du réseau pour éviter ce problème de sur-spécialisation.

T-LESS

YOLO prédit les positions 2D des objets présent dans une image, mais aussi leurs catégories. Nous avons vu les difficultés rencontrées lors de la classification des images de T-LESS à la section précédente. Les objets sont trop similaires et ils comprennent des sous-parties en commun qui ne sont pas retranscrites dans les étiquettes. Nous avons aussi vu la sur-spécialisation qui a lieu sur les images d'entraînements et qui impacte très négativement les résultats sur les images de tests. Les objets étant centrés dans les images d'entraînements de T-LESS, cette sur-spécialisation sera d'autant plus forte pour une méthode de détection 2D comme YOLO. Il est difficile de résoudre ce problème sans utiliser des images d'entraînement synthétiques. Nous rappelons aussi que T-LESS n'est pas un jeu de données conçu pour entraîner des réseaux de neurones.

Pour ces raisons nous n'avons pas testé YOLO sur T-LESS.

YCB VIDÉO

Pour YCB-VIDÉO, nous utilisons la même technique de préparation des images. Nous remplaçons l'arrière-plan de toute l'image en gardant tous les objets d'avant-plan. Puis nous appliquons des changements aléatoires de luminosité, de contraste, de teinte et de saturation.

YCB propose 21 objets différents ($C = 21$). Plusieurs objets différents peuvent être présents simultanément dans une image, mais jamais deux fois le même. Nous utilisons une grille de 5×5 cellules ($S = 5$) avec deux rectangles ($B = 2$) comme dans la version originale de YOLO. Cela permet de bien couvrir les objets présents dans les images de YCB-VIDÉO, même quand les objets sont très proches les uns des autres.

YOLO est entraîné sur YCB-VIDÉO pendant 200 époques avec l'algorithme de descente de gradient stochastique est un taux d'apprentissage de 0.001. Les termes de la fonction de coût décroissent correctement, mais on observe une spécialisation sur les images d'entraînement. On observe dans les images que les plus gros objets sont mieux détectés que les petits. Cette propriété est sans doute due à notre choix de favoriser les grands objets, qui occupe le plus d'aire à l'image, dans le calcul des étiquettes de YOLO. On remarque aussi que YOLO commet beaucoup d'erreur de classification. Prolonger l'entraînement aurait sans doute permis de réduire ces erreurs de classifications. Mais encore une fois, les auteurs avaient fait remarquer que la fonction de coût utilisée pour la classification, la somme des carrés des distances, n'est pas la meilleure pour cette tâche, même si elle permet d'uniformiser l'expression de la fonction de coût de YOLO V1.

Conclusion

YOLO est la première méthode à fournir des résultats de détections en temps réel. Ce gain de vitesse s'effectue au détriment de la précision des résultats par rapport à Mask-RCNN comme les auteurs le disent.

YOLO fonctionne sur les jeux de données que nous avons considérés. Sur LINEMOD, les résultats sont très bons. L'entraînement et les tests du réseau sont simplifiés par le fait qu'un seul objet soit présent dans les images. Déjà, on note que YOLO commet quelques erreurs notamment de classification, comme ses auteurs l'avaient fait remarquer. Sur YCB-VIDEO, les résultats sont encourageants sur les images d'entraînement, mais pas satisfaisants sur les images de tests. Nous aurions pu prolonger l'entraînement, mais cela n'aurait que peu amélioré les résultats.

Pour améliorer les résultats de YOLO V1 sur ces jeux de données, des modifications de notre implémentation sont possibles. Nous n'avons pas implémenté toutes les augmentations d'images utilisées par les auteurs. Il est par exemple possible d'utiliser des rotations, symétries et autres transformations affines d'images pour varier les formes d'objets et de rectangles encadrants. Pour YCB-VIDEO, il nous apparaît comme très important de composer de nouvelles images en "collant" les avants-plans des objets aléatoirement sur des images d'arrière-plan. Pour la fonction de coût, si deux objets "tombent" dans une même cellule, nous affectons l'objet ayant la plus grande aire à l'image. Après coup, il nous paraît important de prendre en compte la visibilité de l'objet. Cela permettrait notamment de privilégier les objets d'avant-plan qui peuvent être moins grands, mais plus visibles.

Enfin, les versions suivantes de YOLO apportent des modifications à la méthode qui sont intéressantes. Les ancres permettent de prédire des rectangles aux dimensions plus précises, tout en permettant de différencier plus efficacement les différents détecteurs de chaque cellule pendant l'entraînement. Le remplacement des couches denses par des couches de convolution 1×1 permet de rendre l'architecture de YOLO complètement convolutionnelle et donc de traiter des images de tailles arbitraires.

Dans le cadre de cette thèse, nous cherchons une position grossière des objets pour pouvoir les recadrer et les passer à une méthode de classification et d'estimation des six degrés de liberté. Cette première implémentation de YOLO permet déjà de localiser grossièrement les objets ce qui est suffisant pour nous.

9.3 Détection 3D

Introduction

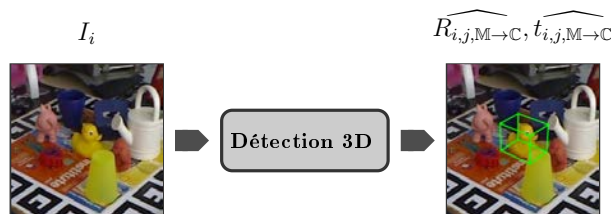


FIGURE 9.3 – Schéma illustrant un module de détection 3D. Les positions $t_{i,j,M \rightarrow C}$ et les orientations $R_{i,j,M \rightarrow C}$ des objets sont estimés.

Pour mettre en œuvre une application de réalité mixte, nous avons besoin de connaître quels objets sont visibles, leurs positions et leurs orientations. L'état de l'art nous a montré que les méthodes les plus adaptées utilisent des réseaux de neurones. Dans cette section, nous avons validé les résultats de plusieurs méthodes. Nous nous sommes efforcés de préparer les images de la même façon pour chaque méthode de façon à pouvoir comparer les résultats.

9.3.1 POSENET

PoseNet [63] fut proposé en 2015 pour déterminer où a été photographiée une image dans un environnement connu. Pour localiser la caméra, le réseau estime les paramètres de pose de la caméra par rapport à la scène. Ce problème est analogue à celui de l'estimation des 6 degrés de liberté d'un objet. Au lieu de chercher la position de la caméra par rapport à l'environnement, nous cherchons la position de certains objets de l'environnement par rapport à la caméra. Nous avons donc décidé d'évaluer les performances de PoseNet pour le problème de l'estimation des 6 degrés de liberté d'un objet. PoseNet modélise la pose d'un objet comme un quaternion pour la rotation (4 paramètres) et un vecteur pour la translation (3 paramètres). La fonction de coût utilisée pour entraîner le réseau est la somme de l'erreur d'estimation de la rotation et de l'erreur d'estimation de la translation.

$$\mathcal{L}(\widehat{q}_{i,j}, \widehat{t}_{i,j}, q_{i,j}, t_{i,j}) = \lambda_q \left\| \widehat{q}_{i,j} - \frac{q_{i,j}}{\|q_{i,j}\|_2} \right\|_2 + \lambda_t \|\widehat{t}_{i,j} - t_{i,j}\|_2 \quad (9.1)$$

PoseNet ne permet pas de reconnaître la catégorie de l'objet dont on cherche la pose.

TLESS Les images d'entraînement de TLESS sont très contrôlées. Et celles de tests particulièrement difficiles du fait des nombreux facteurs de difficultés. Pour commencer, nous réalisons deux séries d'entraînements pour rechercher les valeurs de λ_q et λ_t qui donnent les meilleurs résultats. Nous utilisons seulement les images d'entraînement que nous divisons en deux groupes : entraînement (75 %) et validation (25 %).

Pour la première série d'entraînement, nous avons fixé la valeur de λ_q à 1.0 et varions λ_t . Avec $\lambda_t = 0.01$ la translation a rapidement pris le pas sur la rotation. La précision à 5 cm est très bonne ($\sim 100\%$), mais la précision à 5° est très mauvaise ($\sim 0\%$). Nous avons donc essayé de réduire progressivement la valeur de λ_t pour que PoseNet apprenne la rotation. Pour des valeurs de λ_t de 0.0075, 0.006 et 0.005, la dynamique est toujours la même avec une bonne précision à 5 cm mais une mauvaise précision à 5° . Avec $\lambda_t = 0.005$ on remarque déjà une baisse de la précision à 5 cm sans amélioration de la précision à 5° dans la durée de l'entraînement réalisé. Quand $\lambda_t = 0.001$ alors la rotation prend le dessus et les résultats obtenus sont équivalents à ceux avec $\lambda_t = 0$. λ_t à 0.005 semble être la valeur qui peut donner de meilleurs résultats avec un entraînement plus long et plus d'augmentations d'images. Une valeur de λ_q plus importante pourrait induire PoseNet à apprendre la rotation avec la translation.

Pour la deuxième série, nous fixons la valeur de λ_t à 0.005 et faisons varier celle de λ_q . Nous utilisons les valeurs 1.0, 1.2, 1.5, 2.0 3.0 et 0.5. Lorsque l'on augmente λ_q , alors PoseNet apprend bien la rotation comme le montre le coût sur q qui diminue. Mais il n'apprend pas la translation dans le même temps. Il l'apprend plus tard dans l'entraînement au détriment de la rotation qu'il va oublier.

Par la suite, nous avons réalisé un entraînement plus long : 1000 époques pour lequel toutes les images d'entraînement sont utilisées pour l'entraînement et les images de tests sont utilisées pour la validation et les tests. Nous avons augmenté les images par des changements de luminosité, de contraste, de teinte et de saturation. Nous avons choisi $\lambda_t = 0.005$ et $\lambda_q = 0.75$ car il semble important que PoseNet apprenne la translation avant la rotation qu'il peut oublier. Dans cette configuration, nous espérons qu'avec un entraînement plus long, PoseNet aura le temps d'apprendre la rotation une fois la translation apprise. Nous avons obtenu de bons résultats sur les images d'entraînements. Les précisions à 5° , 5 cm et $5^\circ \wedge 5$ cm sont de 96.31 %, 100 % et 96.31 %. Le recalage 2D à 5 pixels est correct dans 99.99 % des cas et celui en 3D à 5 cm dans 40.95 % des cas. Sur les images de test, les résultats sont mauvais : 3.31 %, 0.65 %, 0.03 %, 0.20 % et 0 % respectivement.

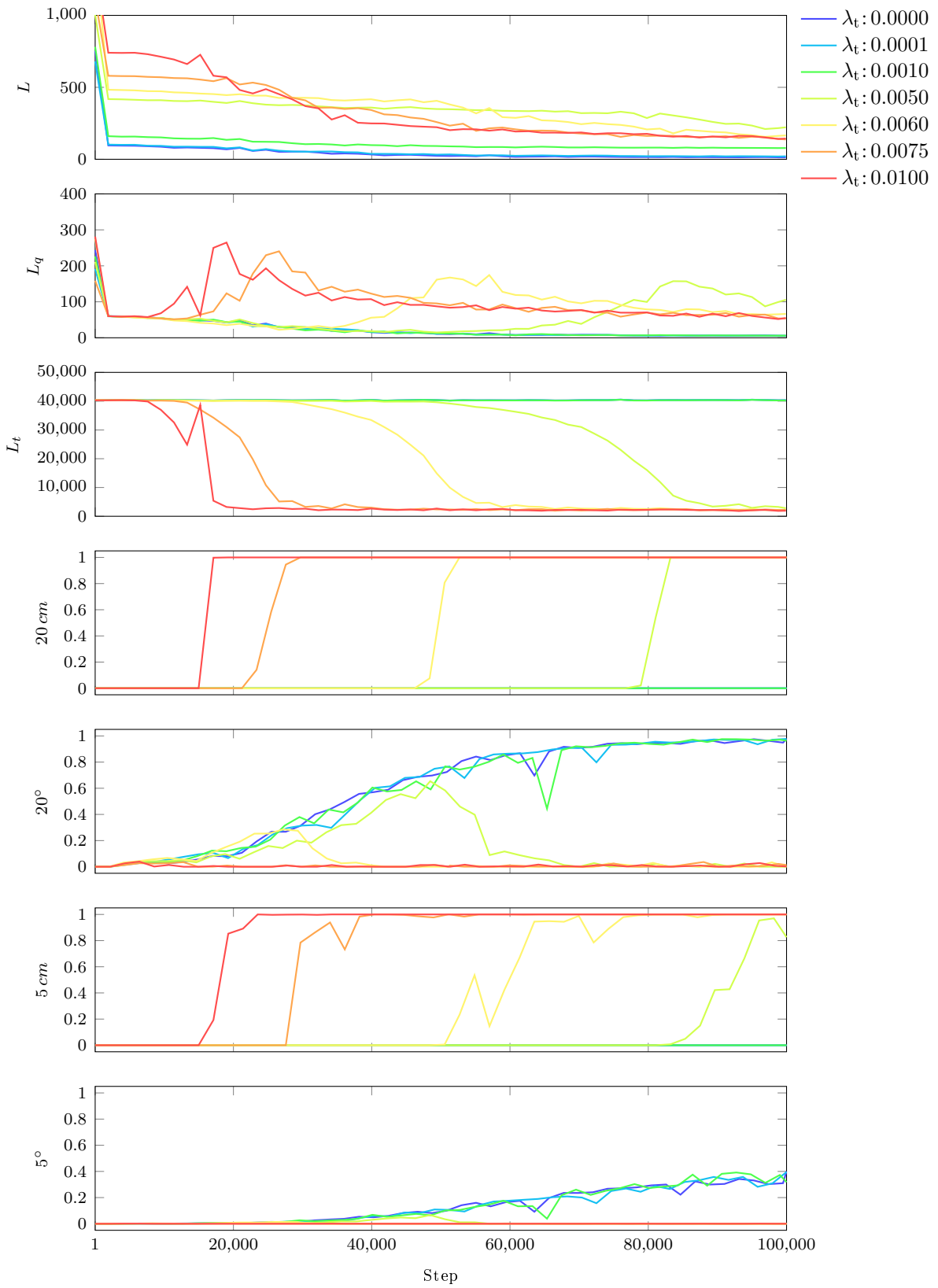


FIGURE 9.4 – Entraînement de PoseNet sur T-LESS avec $\lambda_q = 1.0$ et différentes valeurs de λ_t .

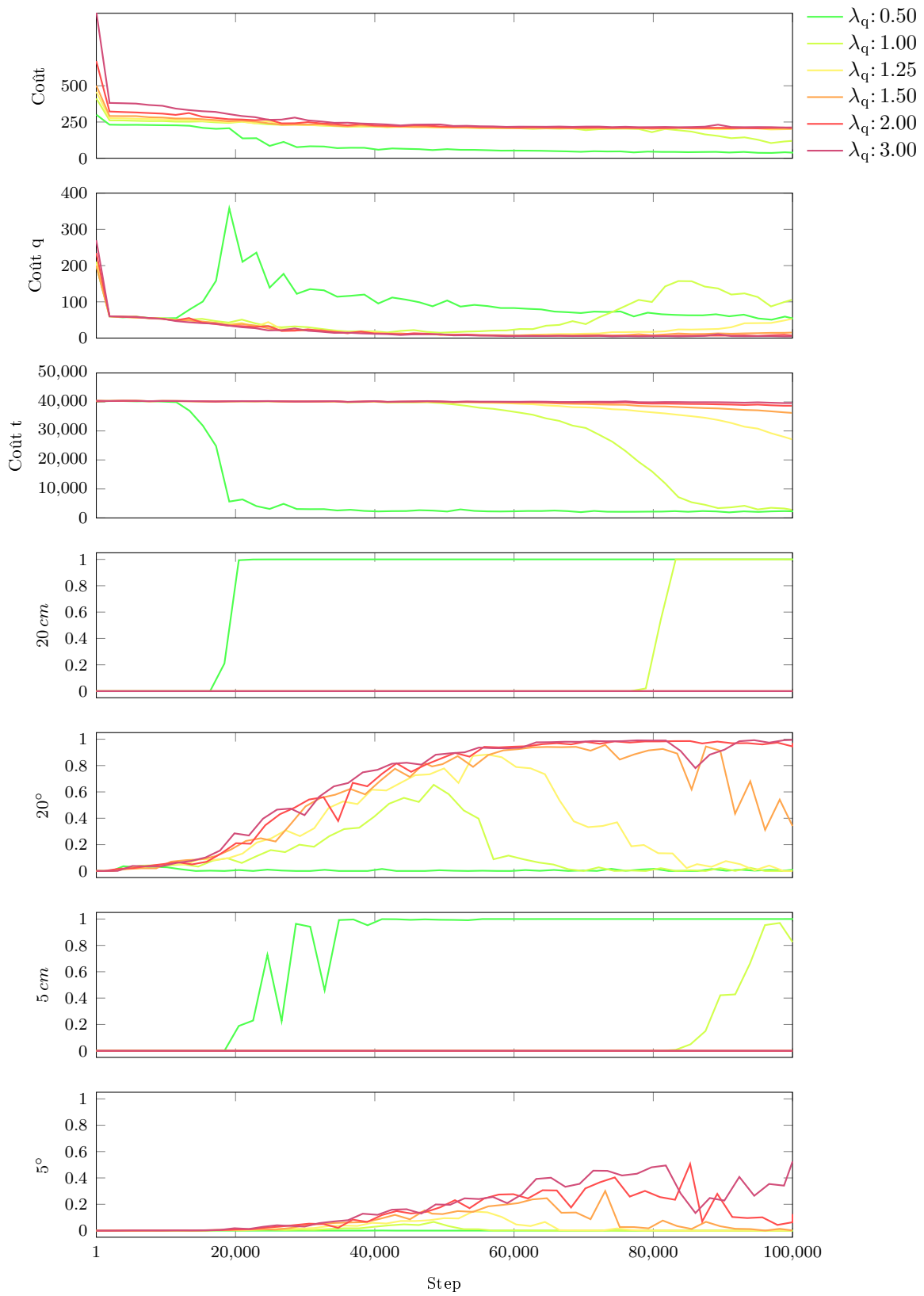


FIGURE 9.5 – Entraînement de PoseNet sur T-LESS avec $\lambda_t = 0.005$ et différentes valeurs de λ_q .

Le problème que nous avons observé lors de notre recherche de λ_q et λ_t est principalement dû au manque d'images d'entraînement et à leur faible variété. En effet, dans les images d'entraînement de T-LESS, la translation varie peu puisque les objets sont toujours centrés et photographiés à la même distance de la caméra. Mais la rotation varie elle beaucoup, puisque les objets sont photographiés de toute part. Il est donc logique que PoseNet se spécialise sur la translation. Le choix de λ_q a donc peu d'importance. Et le choix de λ_t permet seulement d'éviter des valeurs de coût extrêmement élevées sur la translation qui pourraient nuire à l'entraînement. Prolonger l'entraînement et augmenter les images est donc la seule solution qui permet d'entraîner PoseNet sur T-LESS. Malheureusement les images d'entraînement sont trop contrôlées et trop différentes des images de tests pour que PoseNet puisse généraliser.

Ces résultats ont fait l'objet d'une publication à la conférence VISAPP en 2019 intitulée "*Limitations of Metric Loss for the Estimation of Joint Translation and Rotation*" [87] avec la participation de Pascal Desbarats, Jean-Philippe Domenger et Axel Buendia.

LINEMOD Pour entraîner PoseNet sur LINEMOD, nous avons effectué 1000 époques de descente de gradient stochastique avec un taux d'apprentissage fixe de 0.001. Nous pondérons les deux termes de la fonction de coût avec $\lambda_q = 0.500$, $\lambda_t = 0.075$. PoseNet apprend relativement bien à prédire la pose dans les images d'entraînement avec une précision à 5° de 68.38 % et une précision à 5 cm de 56.68 %. Mais la précision des résultats à 5° et 5 cm est limitée avec 41.98 %. Le recalage de la boîte encadrante en 3D est correct à 5 cm dans 56.43 % des cas. Le recalage de la boîte encadrante dans le plan image est extrêmement imprécis avec seulement 2.42 % de boîtes correctement recalées. Sur les images de validation, les résultats sont encore moins bons ce qui montre que PoseNet ne généralise pas bien sur des poses non vues.

Malheureusement, il n'est pas possible d'améliorer ces résultats avec les images à disposition. Nous avons déjà fait varier aléatoirement la luminosité, le contraste, la teinte et la saturation. Nous avons essayé d'appliquer en plus deux autres techniques de recadrage d'images qui avaient donné de bons résultats pour la classification d'images. À savoir recadrer les images aux dimensions des rectangles encadrant, et varier les tailles de ces cadres aléatoirement. Ces augmentations d'images qui varient toutes deux l'échelle dans les images d'entraînement ont toutes deux donné des résultats moins bons. En particulier, c'est la précision de la translation qui s'est effondrée, car c'est la translation qui rapporte la distance à l'objet et donc son échelle dans l'image. Surtout dans le dernier entraînement, où elle n'a pas du tout été apprise. Avec PoseNet, il n'est donc pas possible d'utiliser des augmentations qui déplacent, tournent ou distordent les objets à l'image, car celles-ci sont ambiguës par rapport aux étiquettes décrivant la pose de l'objet par rapport à la caméra.

YCB-VIDÉO PoseNet a été conçu pour retrouver la position d'une image étant donné un ou plusieurs parcours connus, i.e. vues à l'entraînement. Les différentes séquences de YCB VIDÉO correspondent à différents parcours de caméra. De plus, chaque séquence présente des dispositions d'objets différentes. Le protocole de YCB VIDÉO est donc plus compliqué, ce qui devrait mettre PoseNet en difficulté. Nous entraînons PoseNet avec $\lambda_q = 0.500$ et $\lambda_t = 0.075$ pendant 100 époques. Les arrière-plans des images et les distracteurs sont effacés par des images de SUN et nous appliquons des augmentations de luminosité, contraste, teinte et saturation. PoseNet atteint une précision satisfaisante sur les images d'entraînement. La rotation est estimée correctement à 5° dans 88.18 % des images et la translation à 5 cm dans 88.09 %. Ensemble, elles sont correctement estimées dans 79.85 % des images. Le recalage 3D de la boîte encadrante est correct à 5 cm dans 87.97 % des images, mais son recalage en 2D

	LINEMOD		T-LESS		YCB-VIDÉO	
	train	val	train	test	train	test
5°	68.38 %	28.49 %	96.31 %	3.31 %	88.18 %	3.99 %
5 <i>cm</i>	56.68 %	28.80 %	100.00 %	0.65 %	88.09 %	5.02 %
5° 5 <i>cm</i>	41.98 %	11.76 %	96.31 %	0.03 %	79.85 %	0.45 %
3D 5 <i>cm</i>	56.43 %	27.95 %	99.99 %	0.20 %	87.97 %	3.02 %
2D 5 <i>px</i>	02.42 %	0.59 %	40.95 %	0 %	2.39 %	0.01 %

TABLE 9.5 – Résultats de PoseNet.

dans l'image n'est correct à 5 *px* que dans 2.39 % des images. La précision est considérablement inférieure sur les images de test comme nous nous y attendions. Les mêmes mesures que ci-dessus nous donnent 3.99 %, 5.02 %, 0.45 %, 3.02 % et 0.01 % respectivement.

Conclusion sur PoseNet PoseNet est utilisé pour localiser une caméra dans un environnement connu. Les pixels des images qu'il considère appartiennent tous à l'environnement. Ils apportent donc tous de l'information que PoseNet utilise pour localiser la caméra. Lorsque PoseNet est utilisé pour l'estimation des six degrés de liberté, la donne est différente. Seuls les pixels appartenant aux objets apportent de l'information. D'autant plus que nous devons effacer les arrière-plans pour faire disparaître les mires de calibration. PoseNet dispose donc de beaucoup moins de données utiles pour estimer la pose de l'objet. De plus, il doit estimer une pose qu'il n'a pas vue pendant l'entraînement. Sa tâche est donc plus complexe.

Cette complexité se vérifie dans les résultats regroupés au tableau 9.5. PoseNet est capable de recalculer les objets seulement avec une précision de l'ordre d'une dizaine de centimètres. Les résultats sont encore inférieurs quand PoseNet a affaire à des scènes nouvelles comme c'est le cas dans T-LESS ou YCB-VIDÉO. La plus grande limitation de PoseNet est que ces résultats ne peuvent pas être améliorés sans photographier plus d'images réelles ou synthétiques. En effet, toute augmentation d'images qui peut être modélisée comme une modification de la matrice positionnant l'objet par rapport à la caméra est impossible. Cela inclut translations, rotations, symétries, changements d'échelles et déformations affines. Nous pouvons seulement recadrer et redimensionner les images. Avec pour contraintes que le cadre choisi soit le même pour toutes les images, et que les images soient toutes redimensionnées à la même taille. Dans ce cas, ces deux transformations peuvent être intégrées à la matrice de paramètres intrinsèques de la caméra.

9.3.2 BB8

BB8 [91] fait intervenir plusieurs réseaux successivement. Le premier est chargé de segmenter les objets dans l'image. Le second de prédire leurs poses. Et le dernier affine les prédictions étant donné un rendu 3D. Nous avons choisi de ne tester ici que le deuxième réseau consacré à la détection de la pose des objets. Ce réseau prédit les coordonnées de la boîte encadrante d'un objet dans l'image en 2D. Les boîtes encadrantes de différents objets sont prédites simultanément par une seule instance de BB8. Le tenseur en sortie comporte $C \times 8 \times 2$ valeurs pour C catégorie d'objet et 8 points en 2 dimensions. Ce choix a du sens dans le cas de LINEMOD, LINEMOD OCCLUDED ou YCB-VIDÉO, car les objets n'apparaissent jamais plusieurs fois à l'image. BB8 utilise l'ossature de VGG-19 qui d'après les auteurs est plus à même d'apprendre à prédire la pose de plusieurs objets. La seule modification apportée par les auteurs est de modifier le nombre d'unités dans la dernière couche dense et d'utiliser une activation linéaire au lieu du "softmax".

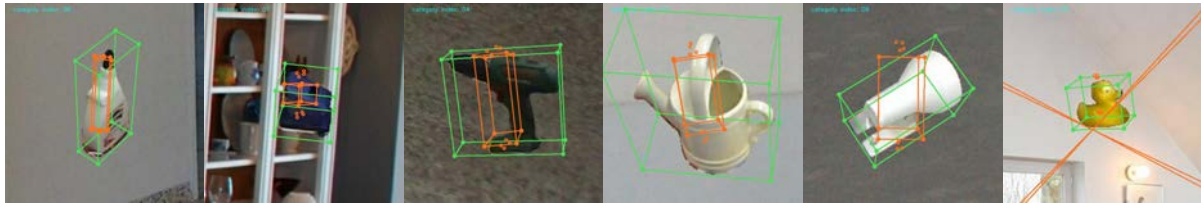


FIGURE 9.6 – Résultats de l’entraînement de BB8 avec des coordonnées en pixels sur LINEMOD. Les boîtes encadrantes obtenues sont effondrées sur elles-mêmes.

LINEMOD Dans l’article original, l’unité des points prédits n’est pas précisée. Nous avons d’abord essayé de prédire directement des points dont les coordonnées seraient exprimées en pixels. La plupart des tentatives ont abouti à un échec de l’entraînement avec un coût augmentant rapidement jusqu’à ce que TensorFlow rapporte une divergence. Si, par chance, lors de l’initialisation, le modèle ne divergeait pas, alors il apprenait des boîtes encadrantes complètement effondrées sur elles-mêmes dont les dispositions étaient relativement constantes. Ce qui semble correspondre à un minimum local de la descente de gradient. Des exemples sont présentés à la figure 9.6.

Les réseaux de neurones ont du mal à modéliser des états prenant des valeurs discrètes. Comme c’est le cas pour la classification d’images où l’on prédit C probabilités plutôt que l’indice de la catégorie d’objet. Par la suite, nous avons donc normalisé les coordonnées des points prédits par BB8 entre 0 et 1 comme dans YOLO [94, 92] et Faster R-CNN [95]. L’entraînement est mené de la même façon que précédemment, mais cette fois-ci, il aboutit. À la fin, les mesures sur les images d’entraînement sont de 39.54 % pour la précision à 5° , 84.19 % pour la précision à 5 cm et 35.50 % ensemble ($5^\circ \wedge 5\text{ cm}$). Le recalage 3D à 5 cm est correct dans 83.73 et celui en 2D à 5 px dans 28.48 %. On remarque aussi que si le recalage en 2D est calculé directement sur les points prédits par BB8 alors il n’atteint que 26.29 %. PnP permet donc un gain de 2.19 % de précision en 2D qui permet aussi d’améliorer les résultats en 3D. Sur les images de validation, les résultats sont similaires bien que légèrement inférieurs : 27.49 % (5 cm), 72.05 % (5°), 22.85 % ($5^\circ \wedge 5\text{ cm}$), 71.20 % (3D 5 cm) et 22.87 (2D 5 px). Nous avons aussi essayé de normaliser les coordonnées des points entre -1 et 1 , les résultats obtenus sont similaires.

T-LESS Nous avons utilisé le même protocole pour entraîner BB8 sur T-LESS. Les résultats obtenus sur les images d’entraînements sont bons en 3D. La précision à $5^\circ \wedge 5\text{ cm}$ est de 84.05 %, le recalage 3D à 5 cm est correct dans 84.05 %. Mais la précision de recalage en 2D n’est pas bonne du tout avec 0.04 %. Ces résultats sont dus aux objets de T-LESS qui comportent beaucoup de symétries et d’ambiguïtés et sont petits. BB8 est capable de les positionner en 3D d’où les bons scores de positionnement et recalage 3D (grâce aux petits objets). Mais il peine à trouver la bonne orientation des objets comme le montre la mesure de précision de la rotation légèrement plus basse et la mesure de recalage 2D extrêmement mauvaise. On remarque aussi que BB8 s’est sur-spécialisé sur les images d’entraînement. Les résultats sur les images de test sont nuls. Les images de tests sont trop différentes des images d’entraînement ne serait-ce que par les poses des objets.

YCB-VIDÉO Pour YCB-VIDÉO, nous filtrons les objets occultés à plus de 70 %. Le reste de la préparation des images est identique. La durée de l’entraînement est de 100 époques comme pour PoseNet. Une fois l’entraînement terminé, BB8 obtient des précisions sur les images d’entraînement de 87.33 % à 5° , 91.65 % à 5 cm , 82.27 % à $5^\circ \wedge 5\text{ cm}$, 91.60 % pour l’ADD 3D à 5 cm et 34.40 % pour l’ADD 2D à 5 px . BB8 apprend donc bien à prédire les

	LINEMOD		T-LESS		YCB-VIDÉO	
	train	val	train	test	train	test
5°	39.54 %	27.49 %	85.44 %	0.12 %	87.33 %	2.12 %
5 <i>cm</i>	84.19 %	72.05 %	97.80 %	0.13 %	91.65 %	15.49 %
5° 5 <i>cm</i>	35.50 %	22.85 %	84.05 %	0 %	82.27 %	0.97 %
3D 5 <i>cm</i>	83.73 %	71.20 %	97.78 %	0.11 %	91.60 %	13.65 %
2D 5 <i>px</i> BB8	26.29 %	20.91 %	0.03 %	0 %	34.40 %	0.79 %
2D 5 <i>px</i> BB8+PnP	28.48 %	22.87 %	0.04 %	0 %	33.17 %	0.86 %

TABLE 9.6 – Résultats de BB8.

coordonnées des coins des rectangles encadrants dans les images d’entraînement. Sur les images de test, les mesures sont bien moins bonnes : 2.12 %, 15.49 %, 0.97 %, 13.65 % et 0.86 % respectivement. BB8 ne généralise donc pas sur les images de test. Les images de test présentent des assortiments d’objets différents et, selon des points de vue, très différents des images d’entraînement, comme nous l’avons vu lors de l’analyse des points de vue proposée par YCB-VIDÉO. Ce qui semble mettre en difficulté BB8 au même titre que PoseNet.

Conclusion sur BB8 Les résultats obtenus par BB8 sont supérieurs à ceux de PoseNet. Sur LINEMOD, on observe un gain de précision de l’ordre 20 % pour l’ADD 2D à 5 pixels par exemple, et de l’ordre de 30 % sur les images d’entraînement de YCB-VIDÉO. Les images de test de YCB-VIDÉO sont problématiques pour BB8 au même titre que PoseNet en raison du caractère inédit des poses et des dispositions d’objets présentés. Le jeu de données T-LESS pose problème à BB8 en raison des symétries et ambiguïtés de rotations propres aux objets de T-LESS. Pour ce jeu de données, les auteurs ont d’ailleurs introduit une nouvelle fonction de coût et une nouvelle mesure de précision qui tient compte de ces ambiguïtés en autorisant les permutations de points lors des calculs.

BB8 soustrait la prédiction de la pose en 3D à PnP. Le réseau de neurones se concentre lui sur la localisation de points dans l’image. Cette stratégie est efficace pour plusieurs raisons. Premièrement, PnP améliore les résultats obtenus, l’algorithme est donc plus efficace qu’un modèle appris pour la même tâche. Deuxièmement, le réseau peut être entraîné avec plus d’augmentations d’image, même si nous avons ici gardé le même protocole que PoseNet par souci de comparaison.

Les résultats détaillés ci-dessus peuvent donc encore être améliorés par plus d’augmentations d’image et une fonction de coût tenant compte de la géométrie des objets ce qui n’était pas possible pour PoseNet.

Chapitre 10

Analyse

Au chapitre précédent, nous avons vu les résultats de détection de plusieurs méthodes de l'état de l'art sur trois jeux de données. Ces trois jeux de données sont très différents comme nous l'avons vu au chapitre de l'état de l'art qui leur est consacré. Mais ces différences nous sont apparues plus clairement lors de la mise en œuvre de nos expériences.

Dans le cas de LINEMOD par exemple, le fait qu'il n'y ait pas d'image d'entraînement nous contraint à trahir le protocole d'utilisation original et à consacrer une partie des images à l'entraînement. Les marqueurs sont visibles dans ces images, ce qui ne pose normalement pas problème lorsqu'elles sont utilisées pour des tests, mais lorsqu'elles sont utilisées pour l'entraînement, il faut effacer les marqueurs. Dans chaque image, seul l'objet au centre du plateau est étiqueté, ce qui pose plusieurs problèmes aussi. Tout d'abord, d'autres objets non labellisés peuvent venir occulter l'objet principal. Lors de nos expériences consacrées aux méthodes de classification d'images (AlexNet, VGG, GoogLeNet), l'objet occultant est correctement reconnu, mais comme c'est l'objet au centre du plateau qui est annoté, le résultat est considéré injustement comme faux. Un exemple est présenté à la figure 10.1. Pendant l'entraînement, ces images ont aussi un impact négatif, car le réseau est pénalisé par la fonction de coût alors que sa prédiction était correcte. Et même si l'image est correctement identifiée pendant l'entraînement, les éléments visuels de l'objet occultant peuvent être associés à l'objet occulté. De plus, le fait qu'un seul objet soit étiqueté pose problème pour les méthodes détectant simultanément plusieurs objets comme YOLO ou BB8. Pour YOLO, nous avons choisi de simplifier la sortie du réseau par n'accommoder qu'un seul objet ($S = 1$), mais le réseau perd la capacité de détecter plusieurs objets. Pour BB8, nous

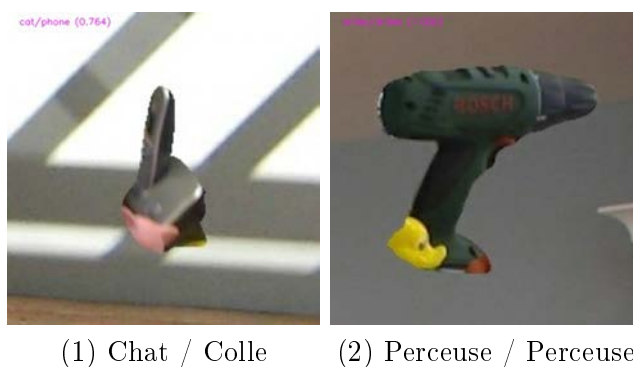


FIGURE 10.1 – Exemple d'images où GoogLeNet classe l'image en "Chat" au lieu de "Colle" car le chat occulte la colle. Dans la deuxième image, la perceuse est correctement identifiée mais, lors de l'entraînement, une telle image peut conduire le réseau à associer certains éléments visuels du canard à la perceuse.

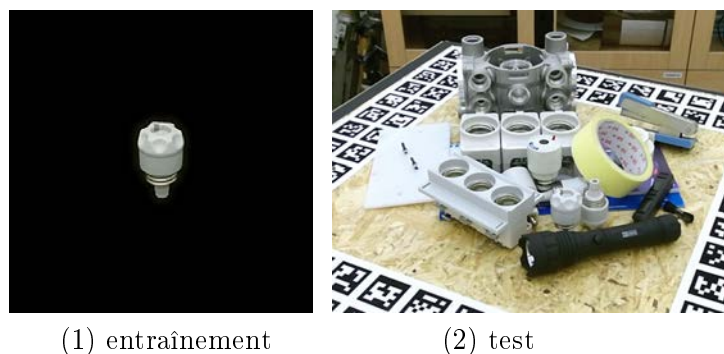


FIGURE 10.2 – Images d’entraînement et de test où apparaissent l’objet 01 de T-LESS. Il est très difficile d’augmenter l’image d’entraînement pour permettre à un réseau de neurones de généraliser sur l’image de test.

n’avons pas modifié le réseau, chaque sortie est donc entraînée séparément image par image. Une autre solution est d’utiliser les avant-plans des objets étiquetés de plusieurs images pour générer de nouvelles images construites en les collant devant un arrière-plan aléatoire. Mais cette implémentation est complexe à réaliser avec TensorFlow sans ralentir considérablement l’entraînement. Cette technique est donc généralement implémentée hors-ligne et réalisée avant l’entraînement, comme le font, par exemple, les auteurs de PVNet. Mais implémenté de cette façon, les images doivent être stockées. On a donc un compromis entre variété des images d’entraînement et espace utilisé pour le stockage des images.

T-LESS a été proposé quelques années plus tard que LINEMOD. Il propose des objets difficiles et des images de tests très complexes, avec plusieurs instances des mêmes objets et des facteurs de difficultés comme des occlusions. C’est un jeu de données très compliqué à utiliser pour entraîner un réseau de neurones, car comme LINEMOD il n’est pas conçu pour cela. Les images d’entraînements ont le mérite d’exister, mais elles présentent un seul objet à une distance fixe de la caméra et sur un fond noir. Les différences marquantes entre images d’entraînement très simples et les images de test très compliquées ont conduit tous les réseaux de détection que nous avons évalués à se sur-spécialiser sur les images d’entraînements. AlexNet et GoogLeNet nous ont montré qu’il est possible de combattre cette sur-spécialisation dans une certaine mesure grâce à des réseaux à l’architecture innovante et, grâce à des augmentations d’images extensives. Mais ces augmentations d’images ne sont pas toutes possibles avec un réseau de détection 3D comme PoseNet, ou elles n’apportent pas le gain de performance espéré comme l’ont montré nos expériences. Cela est aussi dû aux objets du jeu de données qui sont peu différentiables et présentent même des pièces en commun, ce qui n’apparaît pas dans les étiquettes et ne peut donc pas être pris en compte. De ce fait, des rendus 3D sont souvent utilisés dans la littérature pour résoudre ces problèmes.

YCB-VIDÉO est le premier jeu de données destiné à l’entraînement et l’évaluation des réseaux de neurones pour la détection 3D. Plus particulièrement, pour des applications de type préhension robotique. Parmi les objets du jeu de données, les deux pinces sont identiques, modulo un facteur d’échelle. Nos expériences de classification d’images ont montré qu’il faut adresser ces deux objets, soit dans le choix des étiquettes, soit dans la mesure des résultats. Le jeu de données est filmé au lieu d’être photographié. Les objets et leurs poses sont donc échantillonnés le long du parcours de la caméra qui se rapproche des objets à 30 images par secondes. Les exemples sont donc échantillonnés très proches les uns des autres le long du parcours de la caméra dans chaque vidéo. De plus, les objets vus et leurs dispositions ne changent pas au sein d’une même vidéo. Ces deux points font que les images sont très répétitives et qu’il n’y a aucun exemple ailleurs que le long des chemins suivis par

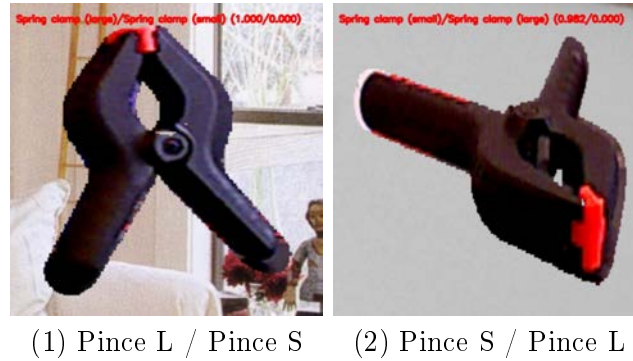


FIGURE 10.3 – Exemples d’images où GoogLeNet confond les deux pinces de YCB-VIDÉO.

la caméra. Nos expériences ont montré qu’il est difficile pour YOLO de généraliser à de nouvelles dispositions d’objets sur ce jeu de données. La seule solution à ce problème semble être de générer de nouvelles images en utilisant les avant-plans des objets comme mentionné ci-dessus. Deuxièmement, les réseaux de détection 3D n’ont jamais vu les objets dans les poses qu’ils prennent dans les images de test. Les poses sont si différentes que PoseNet dépasse à peine les 3% de précision en 3D et atteint seulement 0.01 de précision en 2D. Sans générer plus d’images d’entraînement, BB8 souffre du même problème que YOLO et, en plus, doit prédire des boîtes encadrantes selon des points de vue qu’il ne connaît pas. Il obtient tout de même 13.65% de précision en 3D, mais moins de 0.8% en 2D. Tout cela montre qu’il est très difficile d’utiliser les images de YCB-VIDÉO sans augmentations d’images et sans générer de nouvelles images avec d’autres dispositions d’objets. Pour apporter d’autres images figurant les points de vue inconnus, les auteurs PoseCNN et YCB-VIDÉO ont aussi utilisé des rendus 3D qui semblent être essentiels pour entraîner correctement un réseau de détection 2D ou 3D.

Nous avons réalisé ces expériences car nous souhaitons pouvoir comparer PoseNet et BB8 en terme d’architecture et de modélisation de la pose. Mais, dans tous nos tests, nous avons été limités par les données d’apprentissage, soit absentes du jeu de données, soit présentes mais inconsistantes avec les données de tests. C’est pourquoi nous avons décidé de réaliser un nouveau jeu de données qui adresse les limitations que nous avons rencontrées, et apporte de nouveaux éléments comme des poses plus précises et les arrière-plans réels. La prochaine partie de cette thèse est dédiée à la présentation de ce jeu de données et aux résultats des tests conduits grâce à lui.

Quatrième partie

NEMA

Introduction

WIITY est un logiciel d'aide à la maintenance développé par ITECA que nous souhaitons porter et étendre sur casques de réalités mixtes. Nous souhaitons proposer des augmentations d'objets à nos clients dont la précision serait inférieure au centimètre. Or, comme nous l'avons vu la plupart des jeux de données étudiés ont des problèmes de modèles 3D, car ceux-ci sont reconstruits (LINEMOD) ou manque de précisions quant aux étiquettes (YCB VIDÉO). Les méthodes d'estimation des six degrés de liberté sont donc classées avec des mesures dont les seuils de précision sont de 5 cm et 5° . Certains des jeux de données étudiés n'ont aussi pas été conçus pour l'apprentissage profond (LINEMID, T-LESS). Les images d'entraînement sont, par exemple, toutes photographiées à la même distance de la caméra et peu nombreuses (TLESS), ou des mires de calibration sont visibles (LINEMOD, T-LESS). Les caméras 3D utilisées pour photographier les images dans ces jeux de données ont beaucoup évolué depuis leurs enregistrements. Il y a donc une opportunité pour acquérir un jeu de données avec une meilleure qualité d'images et de cartes de profondeur, des modèles 3D paramétriques et non surfaciques, et sans mire de calibration visible, tout en assurant une qualité irréprochable des étiquettes.

Nous avons donc décidé de saisir cette chance. Nous souhaitons simuler avec notre jeu de données les opérations de montage et démontage d'un moteur NEMA 17. Dans notre scénario, nous imaginons qu'un technicien est assis à son poste de travail. Il porte un casque de réalité mixte qui le guidera dans ses actions. La caméra du casque qui filme la scène est donc légèrement en retrait et surélevée par rapport à la table. Le technicien est concentré et regarde les objets sur lesquels il travaille. Nous avons modélisé et imprimé les pièces du moteur pour pouvoir les disposer de plusieurs façons et ainsi créer plusieurs mises en scènes de montage-démontage où différentes pièces sont disposées de façons variées. Les scènes sont agrémentées avec des objets non annotés pour créer de l'information de contexte.

Les points forts de notre jeu de données sont les suivants :

- La totalité des pixels des images est vue par la même caméra depuis le même point de vue.
- Les marqueurs sont effacés avec l'arrière-plan réel photographié au préalable.
- Les images sont en haute définition 1080×720 .
- Les points de vue sont nombreux et séparés de 0.5° et/ou 0.5 cm .
- Les objets photographiés sont imprimés en 3D à partir de modèles paramétriques.

Ce qui se traduit par 3 contributions :

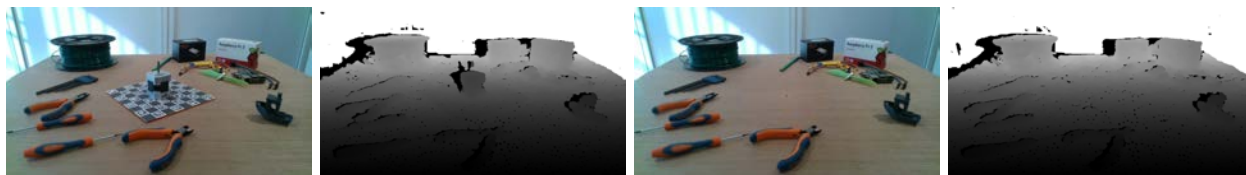
1. Il n'y a donc pas de problème de perspective dans les images, et les informations de contexte et d'échelle sont préservées.
2. Les objets imprimés en 3D sont précisément dimensionnés et leurs modèles 3D n'ont pas de problème de topologie, ainsi les rendus 3D et les masques calculés à partir de ces rendus sont précis au pixel près.



(1) Image couleur où le plateau de marqueurs ChArUco est effacé grâce à la photographie couleur d'arrière-plan.



(2) Carte de profondeur où le plateau de marqueurs ChArUco est effacé grâce à la carte de profondeur d'arrière-plan photographiée.



(3) Avant plan (couleur). (4) Avant plan (profondeur). (5) arrière-plan (couleur). (6) arrière-plan (profondeur).

FIGURE 10.4 – Exemple d'une photographie couleur et d'une carte de profondeur de notre jeu de données NEMA où le plateau ChArUco est effacé.

3. Grâce à la caméra et notre protocole, les images sont de très bonne résolution et les poses sont aussi acquises à une résolution très fine.
4. Il est facile de reproduire ou contribuer à notre jeu de données, car il suffit d'imprimer les objets, le plateau tournant et de posséder la caméra.

Une image illustrant l'effacement du plateau de marqueurs est présentée en figure 10.4. On y remarque le principal avantage de notre protocole d'acquisition qui permet d'utiliser l'arrière-plan réel pour préserver la perspective et les proportions entre les objets d'avant-plan et les objets d'arrière-plan. La carte de profondeur peut être traitée de la même façon, ce qui permet de combler des zones que la caméra ne pouvait pas voir avec les objets à l'avant-plan.

Sommaire

Dans un premier chapitre, nous présentons le matériel utilisé pour la réalisation du jeu de données. Le second chapitre, le protocole à suivre pour acquérir une scène et ses séquences ainsi que le logiciel développé. Le troisième présente les images obtenues et les statistiques du jeu de données. Et enfin le quatrième chapitre présente les résultats des méthodes de détection sur notre jeu de données.

Chapitre 11

Matériel

Ce chapitre présente le matériel que nous avons utilisé pour capturer les images. Dans une première section, nous développons le choix de la caméra. La seconde section apporte des détails sur les objets, leur modélisation et leur impression. La troisième section est consacrée au plateau tournant que nous avons conçu pour les besoins de ce jeu de données. La quatrième et dernière section présente rapidement la table construite pour accueillir le tout.

11.1 Caméra

ITECA développe déjà des applications de réalité mixte sur le casque de réalité mixte Microsoft Hololens. L'entreprise a commandé les Microsoft Hololens 2 dès leur annonce. Le casque avec les outils de développement est facturé 3 500\$. Cette version du casque de réalité mixte de Microsoft embarque une Kinect pour la partie vision. C'est pourquoi nous souhaitons utiliser soit directement l'Hololens 2 monté sur un RIG d'acquisition, soit les Microsoft Azure Kinect DK¹ qui intègrent les mêmes capteurs pour la somme de 400\$. Ou bien plusieurs Kinect puisqu'elles sont capables de fonctionner en groupe.

Malheureusement, la forte demande sur ce matériel combinée à une rupture des stocks, nous a contraint à changer nos plans. Nous avons donc utilisé une Intel Realsense d435i qui est l'équivalent de la Kinect chez Intel. Cette caméra coûte 329\$ ce qui est un peu moins que sa concurrente. Elle embarque une caméra couleur 1080*p*, deux caméras infrarouges d'une résolution maximale de 720*p* et une centrale inertielle. Les capteurs vidéos délivrent un flux vidéo, deux flux vidéos stéréos ou une carte de profondeur. Elle est connectée à l'ordinateur par un câble USB-C 3.1. Un SDK open source C et C++ est développé et mis à disposition par Intel : librealsense². Des versions Python³, C#, Matlab, JavaScript et plus encore sont aussi disponibles. La caméra est aussi supportée dans la bibliothèque OpenCV. La figure 11.1 présente la caméra.

11.2 Objets

Pour les objets, nous voulions qu'ils soient dessinés sur ordinateur et nous souhaitons avoir à disposition les modèles paramétriques. Les modèles paramétriques ne sont pas définis par des sommets et des arêtes, mais par des primitives géométriques et des transformations dont les paramètres sont choisis. Le premier avantage des modèles paramétriques est qu'ils

1. <https://azure.microsoft.com/fr-fr/services/kinect-dk/#overview>

2. <https://github.com/IntelRealSense/librealsense>

3. <https://pypi.org/project/pyrealsense2/>



FIGURE 11.1 – Intel RealSense d425i. Source : intel realsense.

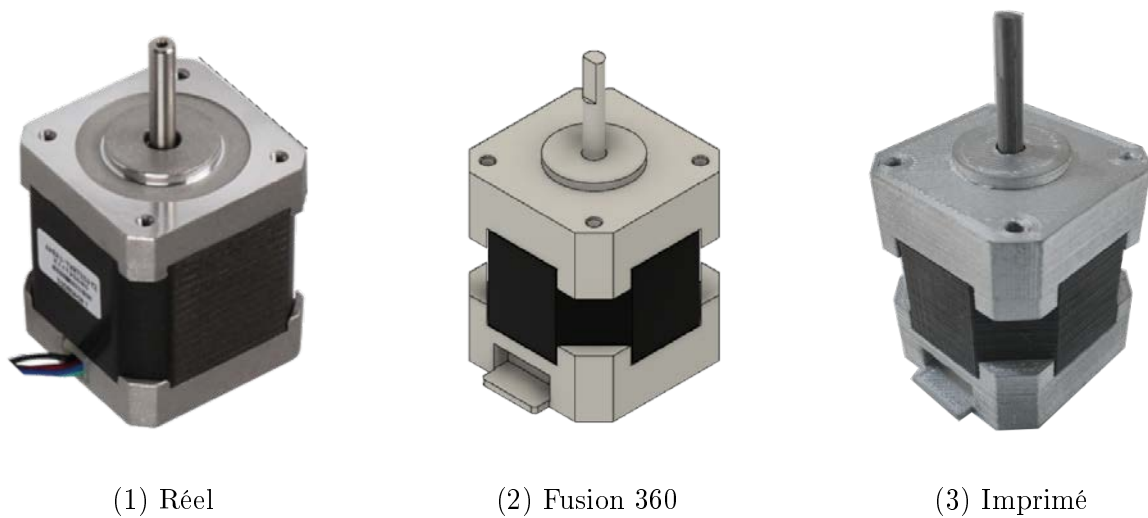


FIGURE 11.2 – Moteur NEMA réel, dessin paramétrique sous Fusion 360 et moteur imprimé utilisé dans notre jeu de données. Source pour l'image du moteur réel : wikipedia.

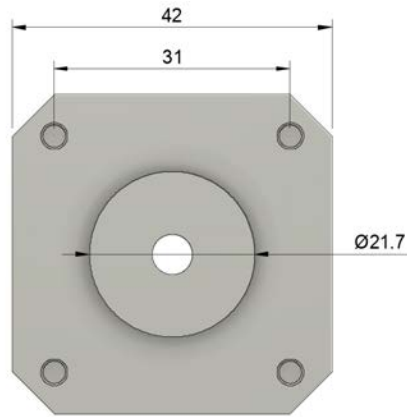


FIGURE 11.3 – Dimensions en millimètres de la façade d'un moteur NEMA 17 tels que définie dans le standard *NEMA ICS 16-2001* [82]

sont souvent plus légers en mémoire. Leur deuxième avantage est qu'ils permettent d'exporter des modèles surfaciques avec différents niveaux de qualité. De ce fait, il n'est pas nécessaire d'utiliser des algorithmes de décimation qui produisent des artefacts.

Nous voulions aussi que les objets aient un caractère industriel pour correspondre à notre cadre applicatif. C'est pour cette raison que nous avons choisi de recréer un moteur NEMA 17 simplifié. Ce type de moteur est illustré à la figure 11.21. L'extérieur du moteur est recréé avec fidélité en respectant les dimensions définies par la norme "*NEMA ICS 16-2001*" [82]⁴. Ces dimensions sont présentées à la figure 11.3. Nous n'avons pas recréé l'intérieur avec le bobinage. Le dessin, réalisé avec Fusion 360, est présenté à la figure 11.22. Le standard qui définit les dimensions de ce moteur ne s'intéresse qu'aux dimensions de la plaque de montage. De ce fait, selon les marques, les fabricants et les utilisations, le reste du moteur est susceptible de changer. Par exemple, la hauteur du moteur varie en fonction de la puissance du moteur, car elle permet d'augmenter la taille du bobinage et des aimants et donc le couple du moteur. Pour cette raison, nous n'avons pas filmé un moteur réel, mais une impression 3D de notre propre dessin. Le résultat après impression, ponçage et peinture est présenté en figure 11.23. En faisant cela, nous garantissons que toute personne qui dispose d'une imprimante 3D, de filament PLA noir et d'une bombe de peinture métallisé puisse reproduire exactement notre objet et à bas coût.

De plus, la question du transfert d'apprentissage entre reproduction imprimée en 3D et objet réel est une question qui n'a pas été abordée à notre connaissance dans le domaine de la vision par ordinateur. Alors que les maquettes et les jouets sont utilisés depuis très longtemps pour l'enseignement chez l'Homme.

11.3 Plateau tournant

Après l'étude des jeux de données existants, il est clair qu'un plateau tournant en combinaison avec des marqueurs ChArUco est la clé pour obtenir de nombreuses images avec une annotation précise. Des nombreuses solutions de plateaux tournants sont proposées à la vente. Mais les plateaux de grandes dimensions pour la réalisation de photographies sont coûteux. De l'ordre de 130€ pour un plateau de 65 cm.

Pour ne pas introduire d'élévation entre nos objets et la table, nous avons besoin d'un plateau aussi fin que possible. Sinon, une fois le plateau effacé dans les images, cela donnerait

4. Voir les pages 50-52, section 4.1.2.3 : dimensions for mounting flanges.

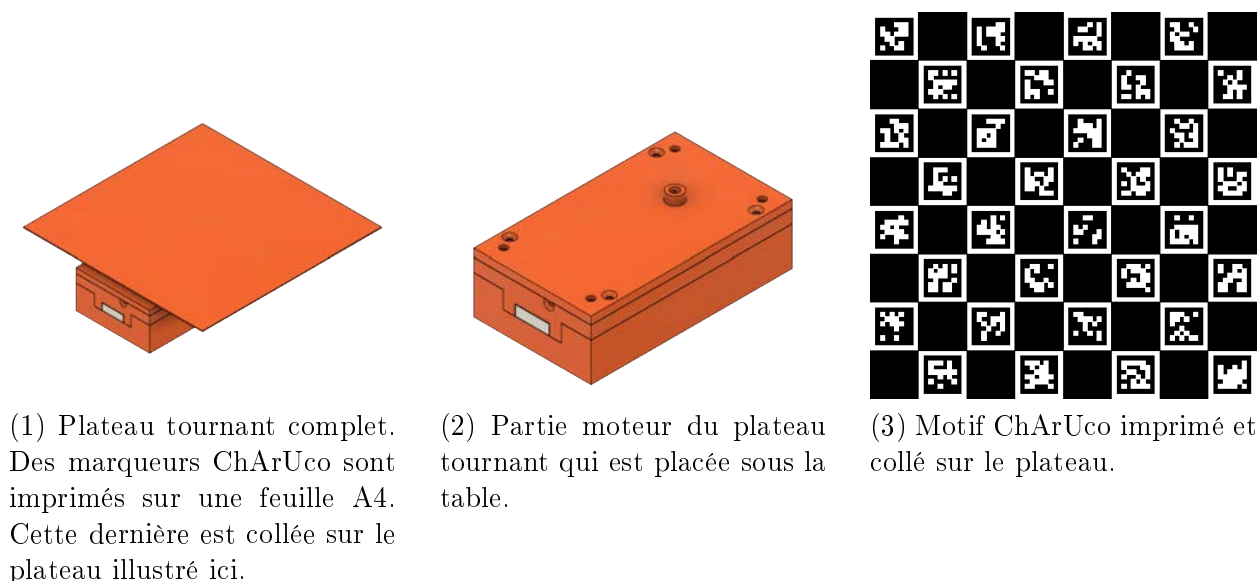


FIGURE 11.4 – Plateau tournant conçu et utilisé pour réaliser notre jeu de données.

l'impression que les objets flottent dans l'air. Or, les solutions commerciales sont épaisses, car elles intègrent le moteur, le microcontrôleur et l'alimentation. Presque 7 cm pour le même plateau que précédemment.

Nous avons donc conçu un plateau dont le moteur est dissimulé sous la table. Le microcontrôleur utilisé est un Arduino Nano Every⁵ (11€). Il est lui-aussi déporté pour alléger la construction et pilote un moteur pas-à-pas 28BYJ-48⁶ ($\sim 4\text{€}$). Le plateau amovible, d'une épaisseur de $\sim 1\text{ mm}$, est accouplé au moteur par un trou de 10 mm dans la table. L'objet est placé au centre du plateau, à la verticale du trou, et dissimule donc le trou dans toutes les images, même après avoir effacé le plateau.

11.4 Table et support caméra

Pour la table et le support de la caméra, nous avons utilisé une combinaison de tubes profilés en aluminium avec des plaques de contre-plaqué. La table est un simple cube de 120 cm de côté et 80 cm de hauteur. La caméra est soutenue verticalement par un tube et sa hauteur peut être ajustée manuellement. Ce tube peut lui-même coulisser horizontalement pour rapprocher ou éloigner la caméra de la table. Nous avons placé sur ces deux tubes des graduations en collant un mètre ruban pour pouvoir positionner la caméra à espaces réguliers. Ce système ne participe en rien au calcul des étiquettes, c'est juste un repère pour la personne faisant l'acquisition des images. Notre support de caméra est comparable à celui utilisé pour T-LESS.

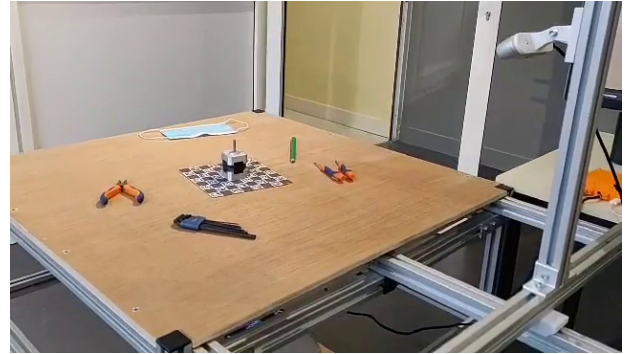
Nous aimerions faire évoluer ce support pour limiter encore les opérations manuelles de placement de la caméra.

5. <https://store.arduino.cc/collections/boards/products/arduino-nano-every>

6. https://www.amazon.fr/Kuman-ULN2003-Arduino-28BYJ-48-Conducteur/dp/B01IP7I0GQ/ref=sr_1_5



(1) Table dans son ensemble.



(2) Caméra sur son support et exemple de mise en scène.

FIGURE 11.5 – Table d’acquisition utilisée pour photographier notre jeu de données. La partie de gauche est utilisée pour mettre en scène les objets. Celle de droite accueille le PC utilisé pour l’acquisition.

11.5 Environnement

La salle que nous utilisons pour nos expériences est située au rez de chaussé de notre laboratoire. Elle est exposée plein sud et dispose d’une grande baie vitrée. Elle reçoit donc généreusement la lumière le matin. De ce fait, nous avons évité de filmer pendant les heures matinales pour limiter les ombres sur la table et la surexposition des images. Les nombreux spots LED permettent d’homogénéiser l’éclairage quand nous avons photographié les images.

Pour agrémenter les mises en scènes, nous disposons d’autres objets sur la table autour du plateau. Nous nous assurons que ces objets ne soient pas placés entre la caméra et le plateau ou les objets d’avant-plan. Ceci pour éviter toute occlusion qui n’apparaîtrait pas dans les étiquettes ou qui générerait la détection du plateau ChArUco.

Conclusion

Nous avons choisi d’utiliser beaucoup de solutions "fait maison". Celles-ci sont, dans notre cas, moins chères, plus simples à mettre en œuvre, tout aussi simples à utiliser, et répondent mieux à nos besoins. De plus, cela les rend abordables pour tous, puisqu’elles peuvent être facilement reproduites moyennant une imprimante 3D.

Chapitre 12

Protocole d'acquisition

L'acquisition de notre jeu de données est découpé en 4 étapes :

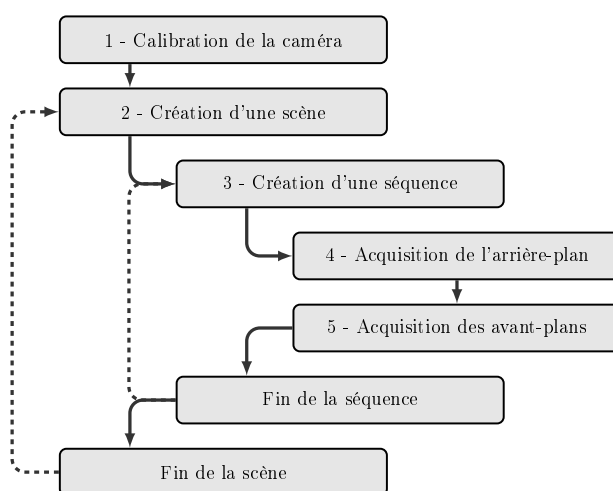


FIGURE 12.1 – Schématisation du protocole d'acquisition.

Ce découpage permet de faire l'acquisition de données de façon à tirer pleinement parti du plateau tournant. Il permet de maintenir la perspective avec l'arrière-plan qui est acquis à l'étape 4. Il permet aussi de limiter au maximum les annotations manuelles puisque la position des objets est saisie une seule fois par scène comme nous allons le voir.

12.1 Calibration de la caméra

Tout d'abord, la caméra est initialisée et ses paramètres intrinsèques sont définis dans notre application. S'ils ne sont pas connus, nous avons mis en place un outil de calibration pour les calculer à partir d'une séquence d'images. Cet outil se base sur l'implémentation des marqueurs ChArUco dans la bibliothèque OpenCV. Pour réaliser la calibration, il faut tout d'abord placer le plateau ChArUco sur la table. On positionne la caméra et on s'assure que le plateau ChArUco est complètement dans l'image. Puis, on lance l'acquisition d'une série d'images pendant que le plateau tourne. On déplace la caméra et répète l'acquisition quelques fois pour varier les points de vue. Les bibliothèques OpenCV et ChArUco sont ensuite utilisées pour détecter les coins des marqueurs et des cases du plateau dans chaque image. Puisque la géométrie du plateau est connue, la bibliothèque est ensuite capable de calculer les paramètres intrinsèques à partir des coins détectés dans toutes les images grâce à PnP.

Nous avons effectué la calibration de notre caméra et les valeurs obtenues sont très proches de celles données par le constructeur :

Paramètre	Valeur
point principale	(636.2532958984375, 356.8138427734375)
longueurs focales	918.0423583984375, 916.5652465820312
résolution	1280×720
coefficients de distordions	0, 0, 0, 0, 0
échelle de profondeur	1.0000000474974513

TABLE 12.1 – Paramètres intrinsèques de la caméra Intel Realsense d453i.

12.2 Création d’une scène

Lors de la création d’une scène, le plateau ChArUco est placé sur la table. Des objets sont disposés à sa surface. Par exemple, la base et le corps du moteur sont assemblés, mais pas la partie supérieure. La position et l’orientation de chaque pièce est ensuite manuellement annotée par rapport au plateau. Pour nous aider, nous avons utilisé un plateau ChArUco dont les cases font la moitié de la base de notre objet, soit 21 mm . Nous avons développé une interface qui permet d’importer les modèles 3D des objets présents dans la scène et d’annoter leurs poses. De façon à limiter l’erreur, ce logiciel affiche un rendu des modèles 3D des objets par dessus le flux vidéo de la caméra. La couleur et l’opacité des modèles peuvent être modifiées pour une meilleure visualisation de la pose. Il est aussi possible de faire tourner le plateau pour vérifier que l’annotation est bonne sous différents points de vue. Après cette étape, on retrouve dans le jeu de données les modèles 3D, des pièces utilisées dans la scène et leurs dispositions par rapport au plateau ChArUco. La disposition est sauvegardée pour chaque pièce comme la transformation affine permettant de passer du repère de la pièce à celui du plateau ChArUco. Ces étiquettes sont communes à toutes les séquences et images de la scène.

12.3 Création d’une séquence

Une fois la mise en scène terminée, on retire tous les objets du plateau. La caméra est placée à une certaine distance et élévation par rapport à la table. Par exemple 15 cm de recul et 30 cm d’élévation. Ce positionnement est effectué manuellement en faisant glisser le support de la caméra le long des supports. La caméra est ensuite orientée de façon à ce que le centre de l’image coïncide avec le centre du plateau. Cette étape nous permet d’assurer un biais central dans nos images. La position de la caméra ne change plus pour le reste de la séquence. Pour agrémenter les séquences, nous ajoutons d’autres objets non annotés autour du plateau. Cela permet de varier la composition des images et d’enrichir les images couleur et les cartes de profondeur avec plus de détails. Ces objets sont différents des objets annotés et nous nous assurons qu’ils n’occulent pas le plateau ni les objets d’avant-plan. Il est impossible d’identifier ces occlusions après coup dans les étiquettes, il est donc essentiel de s’assurer qu’il n’y en ait pas.

12.4 Acquisition de l'arrière-plan

Avant de capturer l'arrière-plan, on retire le plateau et les objets annotés de façon à ce que seul l'arrière-plan soit visible. Dans l'arrière-plan seront donc visible la table, les objets non annotés et les murs de la salle où nous avons fait l'acquisition.

Une fois l'acquisition faite, on obtient la photographie couleur et la carte de profondeur qui sont stockées comme deux images PNG. L'alignement entre ces deux images est assuré par la bibliothèque logicielle "librealsense" fournie par le fabricant de la caméra. L'image couleur contient trois canaux de 8 bits par pixel, soit 24 bits par pixel au total. La carte de profondeur est monocanal, mais avec 16 bits par pixel. En multipliant la valeur des pixels de la carte de profondeur par l'échelle de profondeur ("depth scale"), on retrouve une carte de profondeur avec des valeurs flottantes dont l'unité est le mètre (c.f. 12.1).

12.5 Acquisition des avant-plans

La dernière étape consiste à remettre le plateau et les objets en place sur la table tels qu'ils étaient lors de la création de la scène. Le plateau s'insère dans sa base et se positionne sans effort. Pour réaligner les pièces sur le plateau, on peut s'aider de notre outil qui affiche les modèles 3D et permet de faire tourner le plateau. Cette étape doit être réalisée avec soin. Enfin, on lance l'acquisition de la séquence en appuyant sur un bouton "photo" dans l'interface utilisateur. Le plateau tourne par pas successifs dont on peut choisir l'angle. À chaque pas, une photographie couleur et la carte de profondeur associée sont capturées. Les marqueurs ChArUco, les coins du plateau et la pose du plateau sont automatiquement détectés et calculés grâce à la bibliothèque OpenCV et ArUco. Puis, par l'application des transformations affines définies lors de la création de la scène, la pose de chaque pièce est déduite de celle du plateau. De cette acquisition résulte une séquence d'images et de cartes de profondeur et leurs étiquettes. Les étiquettes contiennent la liste des pièces présentes identifiées par leur index et leur nom ainsi que leur translation et rotation par rapport à la caméra. Si l'estimation de la pose du plateau ChArUco échoue, alors la capture des avant-plans est interrompue. L'estimation échoue quand la caméra rase trop la table ou qu'elle se trouve trop loin du plateau et que les cases du plateau deviennent difficiles à discerner. Dans ce cas, nous supprimons la séquence et passons à la suivante.

Une fois les avant-plans acquis, la séquence est complète. On peut alors créer une autre séquence pour la même scène, auquel cas on retourne à l'étape 3. Ou on peut créer une nouvelle scène, et on retourne à l'étape 2.

12.6 Comparaison

Dans LINEMOD et YCB-VIDÉO, la caméra se déplace par rapport aux objets ce qui ne permet pas de photographier les arrière-plans qui seraient différents pour chaque image acquise. Dans T-LESS, le plateau tournant n'est pas amovible et ne permet pas aux auteurs de photographier l'arrière-plan avant de lancer l'acquisition d'une scène. Notre protocole reprend celui de T-LESS, tout en lui apportant plus de granularité. Ce qui, en plus d'un plateau amovible, nous permet de photographier les arrière-plans de chaque séquence.

En contre-partie, notre protocole nécessite plus d'annotations manuelles. Il faut aligner manuellement les objets deux fois : une fois lors de la création de la scène et une fois après avoir photographié l'arrière-plan d'une séquence. C'est plus fastidieux que LINEMOD, T-LESS, ou YCB-VIDÉO où l'alignement n'est réalisé qu'une seule fois par scène. Mais

c'est la seule façon de garantir une bonne qualité d'étiquette comme nous allons le montrer dans le chapitre suivant. Dans T-LESS, les étiquettes sont automatiquement corrigées après l'acquisition d'une scène. Nous n'avons pas implémenté cette étape, car nos étiquettes sont déjà d'une qualité suffisante, mais cette étape peut être utile si plus de personnes sont amenées à faire l'acquisition du jeu de données.

Dans le chapitre suivant nous présentons le jeu de données dont nous avons fait l'acquisition à travers quelques statistiques, dont la qualité des étiquettes.

Chapitre 13

Jeu de données

Nous avons fait l'acquisition de 6 scènes. La première met en scène le moteur assemblé et propose un échantillonnage des poses très fin. Nous appelons cette scène "HD" pour "haute définition". Les 5 autres correspondent à différentes étapes d'un scénario de montage du moteur.

13.1 Scène HD

Dans la scène HD, le plateau tourne avec un pas de 0.5° soit 720 images pour faire un tour complet. La caméra est déplacée centimètre par centimètre le long de son support. Nous arrêtons de reculer la caméra lorsque le plateau n'est plus détecté correctement. Au total, nous avons capturé 172 800 images dans lesquelles les quatre pièces sont visibles, soit 691 200 instances d'objets. Comme pour les jeux de données de l'état de l'art, nous détaillons ici les points de vue, les positions des objets à l'image, la visibilité des objets et la qualité des étiquettes.

13.1.1 Points de vues des objets

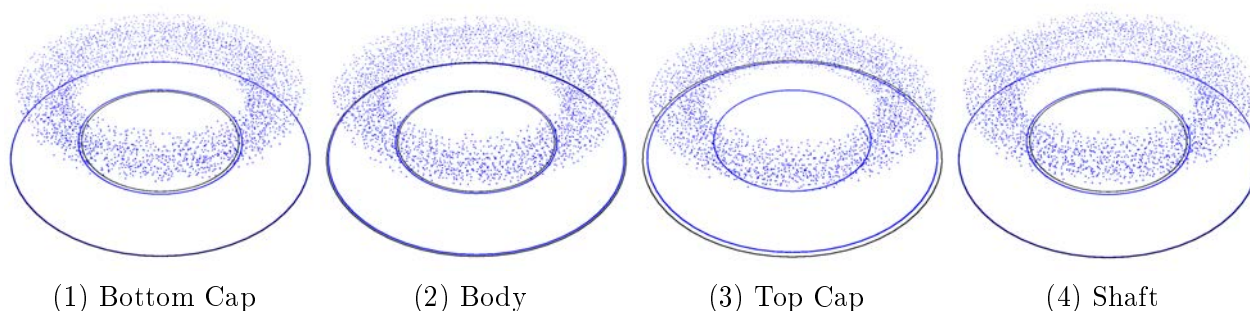


FIGURE 13.1 – Captured points of views of NEMA parts in the first scene.

L'espace des points de vue prend la forme d'un donut comme le montre la figure 13.1. On remarque la densité des points de vue capturés (alors même que la figure 13.1 les sous-échantillonne pour alléger ce document PDF). La forme de l'espace occupé par les points de vue est dû au fait que de trop loin le plateau n'est plus détecté correctement. Nous sommes donc limités en recul par rapport au plateau. Ce problème peut être résolu en utilisant des cellules et des marqueurs plus grands sur le plateau, ou pourquoi pas une combinaison de petites et de grandes cellules. L'espace des poses est surélevé par rapport à la table, car le

plateau n'est pas détecté non plus si la caméra a une vue trop rasante. Dans T-LESS, ce problème est résolu en plaçant des marqueurs sur le bord du plateau, ce que nous ne pouvons pas faire étant donné que nous voulons un plateau le plus fin possible.

Notre jeu de données met en scène l'assemblage ou la maintenance d'un moteur par une personne supposée être assise à la table. La personne étant assise, sa tête se trouve donc à une distance de l'ordre du mètre de la table et surélevée par rapport à celle-ci. Elle ne rapprochera normalement pas sa tête au ras de la table et, si elle s'éloigne trop, on considère qu'elle sort du cadre. Les points de vue que nous avons capturés sont donc cohérents avec ce scénario.

13.1.2 Position des objets à l'image

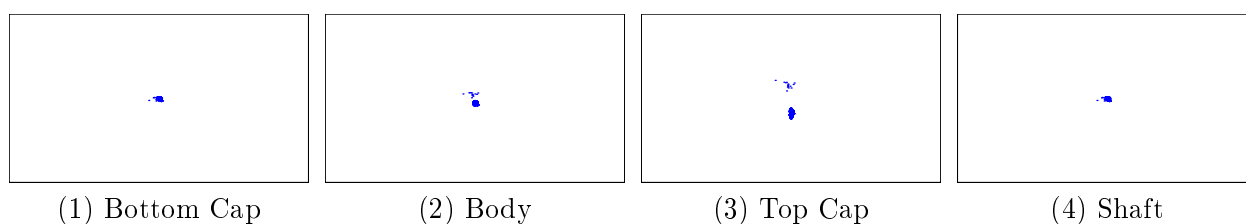


FIGURE 13.2 – Localisation des objets dans les images de NEMA

Comme expliqué dans le chapitre précédent, nous faisons une mise au point manuelle de la caméra pour créer un biais central dans les images. Les objets sont donc centrés dans les images comme le montre la figure 13.2. Ils sont seulement décalés verticalement dans l'image selon leur agencement lors de mise en scène.

Le biais central est important pour deux raisons. Premièrement, il simule la saillance visuelle Humaine [47, 20, 88] naturellement présente dans les images qui seront acquises par le casque de réalité mixte. Il y aura une saillance dans les images, car le technicien regardera forcément les objets sur lesquels il travaillera. Deuxièmement, nous souhaitons seulement évaluer les performances d'estimations des six degrés de liberté des objets et non leurs localisations dans le plan image.

13.1.3 Visibilité des objets

Pièce(s)	$\min(v)$	$\max(v)$	$\mu(v) \pm \sigma(v)$
Body	35.20 %	60.76 %	46.76 ± 04.35 %
Bottom Cap	35.26 %	61.51 %	48.33 ± 04.33 %
Charuco	81.84 %	87.87 %	84.78 ± 01.36 %
Shaft	32.18 %	41.19 %	36.67 ± 01.09 %
Top Cap	95.38 %	98.44 %	96.77 ± 00.46 %
Toutes	32.33 %	98.44 %	62.72 ± 23.80 %

TABLE 13.1 – Visibilité minimum, maximum, moyenne et écart type des pièces du jeu de données NEMA.

Les pourcentages de visibilité sont présentés au tableau 13.1. La scène HD comprend une seule disposition des pièces. Les visibilités de ces pièces sont donc relativement les mêmes dans toutes les séquences de la scène. De ce fait, pour une même pièce, la variance ne dépasse pas les 4.35%. Sans surprise, l'objet le plus visible est le capot supérieur ("top cap") avec

96.77 % qui couvre les autres objets. Le deuxième objet le plus visible est le plateau ChArUco avec 84.78 %. Le plateau est modélisé comme n'importe quelle pièce et sa pose est connue dans nos étiquettes. Les autres pièces sont entre 36.67 % et 46.76 % visibles ce qui est cohérent avec la géométrie des pièces.

Ces visibilités permettent de tester les performances d'une méthode d'estimation des 6 degrés de liberté face aux occlusions partielles. On note qu'aucune pièce n'apparaît hors champ ou n'est complètement occultée. Il n'est donc pas nécessaire de filtrer les instances d'objets de notre jeu de données en fonction de leurs visibilités.

13.1.4 Qualité des étiquettes

Pièce(s)	$\mu(\delta_p) \pm \sigma(\delta_p)$	$\mu(\delta_p) \pm \sigma(\delta_p)$	Ignorées
Body	$-2.2747 \pm 3.6766 \text{ mm}$	$3.1086 \pm 3.0046 \text{ mm}$	3.84 %
Bottom Cap	$-0.6711 \pm 4.8036 \text{ mm}$	$2.6869 \pm 4.0381 \text{ mm}$	3.67 %
Shaft	$0.8189 \pm 1.8361 \text{ mm}$	$1.5271 \pm 1.3077 \text{ mm}$	30.07 %
Top Cap	$1.1433 \pm 4.1621 \text{ mm}$	$3.1636 \pm 2.9363 \text{ mm}$	11.65 %
Charuco	$1.3696 \pm 2.5955 \text{ mm}$	$2.0787 \pm 2.1896 \text{ mm}$	2.63 %
Toutes	$1.0899 \pm 3.0482 \text{ mm}$	$2.2352 \pm 2.3417 \text{ mm}$	10.37 %

TABLE 13.2 – Erreur de recalage des pièces de NEMA.

Comme pour les autres jeux de données, nous mesurons l'erreur de recalage des modèles 3D dans les cartes de profondeur. Les résultats sont détaillés au tableau 13.2. Notre jeu de données obtient une erreur plus faible que tous les jeux de données de l'état de l'art. L'erreur moyenne de recalage est de 1.0899 mm , et 2.2352 mm en valeur absolue. Pour ces deux moyennes, les écarts types sont aussi inférieurs à ceux des autres jeux de données.

On remarque que le plateau ChArUco est très bien recalé et qu'il présente peu de pixels ignorés (2.63 %). Le plateau est donc un bon référentiel de base à partir duquel calculer la pose des autres objets. La base du moteur (bottom cap) et son corps (body) sont toutes les deux très bien recalées avec $\sim 3 \text{ mm}$ d'erreur et moins de 4 % de pixels ignorés. La partie supérieure du moteur (top cap) est bien recalée avec 1.14 mm d'erreur. Elle présente 11.65 % de pixels ignorés, ce qui est plus que les deux pièces précédentes. Cette erreur est probablement due au jeu laissé entre les pièces pour pouvoir facilement les assembler. Le jeu peut introduire un décalage qui s'accumule de la base à la partie supérieure du moteur. L'axe du moteur (shaft) est très bien recalé avec 0.81 mm d'erreur, mais beaucoup de pixels sont ignorés : 30.07 %. Ces pixels ignorés sont dus à des problèmes d'acquisition de l'axe dans les cartes de profondeur et non à un problème d'étiquette. C'est notamment visible à la figure 10.4. L'axe est gris et fin ce qui semble poser problème à la caméra, surtout dans la partie basse près de la partie supérieure du moteur. Ces pixels sont donc ignorés à juste titre du calcul d'erreur puisque dus à une erreur d'acquisition et non d'étiquetage.

13.2 Autres Scènes

Nous avons aussi enregistré cinq autres scènes avec un angle et un pas plus grand entre chaque image. Elles sont donc composées de moins de séquences par scène et d'images par séquence. Les trois premières mettent en scène les pièces utilisées dans la scène HD, mais individuellement (base, corps et partie supérieur). Les deux scènes suivantes sont les premières étapes de l'assemblage du moteur avec l'ajout du roulement à billes à la base du moteur

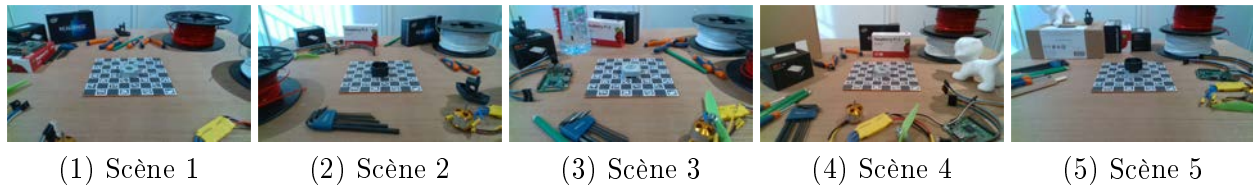


FIGURE 13.3 – Présentation des autres scènes du jeu de données où figurent des pièces seules ou des assemblages partiels du moteur.

(scène 4), puis l’ajout du corps du moteur (scène 5). Dans cette dernière scène, le roulement à billes est souvent complètement occulté par le corps du moteur.

13.3 Comparaison

	LINEMOD	OCCLUDED	T-LESS		YCB-VIDÉO		NEMA
			train	test	train	test	
$\mu(v) \pm \sigma(v)$	96.47 ± 3.68 %	79.31 ± 24.23 %	100 %	83.06 ± 26.09 %	86.17 ± 19.94 %	89.19 ± 15.65 %	62.72 ± 23.80 %
hors champs	0 %	0.06 %	0 %	0 %	0.49 %	0 %	0 %
$\mu(\delta_p) \pm \sigma(\delta_p)$	1.49 ± 7.75 mm	3.27 ± 7.52 mm	5.36 ± 10.65 mm	3.02 ± 12.15 mm	1.11 ± 5.09 mm	0.98 ± 4.59 mm	1.09 ± 3.05 mm
$\mu(\delta_p) \pm \sigma(\delta_p)$	5.07 ± 6.05 mm	5.54 ± 6.05 mm	7.97 ± 08.87 mm	8.97 ± 08.74 mm	2.80 ± 4.39 mm	2.68 ± 3.85 mm	2.24 ± 2.34 mm
ignorés	7.30 %	12.15 %	5.46 %	7.74 %	20.60 %	19.38 %	10.37 %

Bien que notre protocole demande des annotations manuelles, il nous a permis d’acquérir un jeu de données plus vaste et plus précis que les jeux de données de l’état de l’art. Rien que la scène HD comprend plus d’images que YCB-VIDÉO qui fut le premier jeu de données à fournir suffisamment d’images pour entraîner un CNN directement. Nos étiquettes sont aussi plus précises que tous les autres jeux de données non seulement en terme d’erreur moyenne de recalage, mais aussi en pourcentage de pixels ignorés. Cette précision est due à notre protocole, le soin apporté au positionnement des objets lors de l’enregistrement, et à la caméra utilisée qui est plus récente et plus précise. Cette caméra nous permet d’ailleurs d’acquérir les images couleur et cartes de profondeur en 720p faisant de notre jeu de données le premier avec des images haute définition par sa résolution d’images. Grâce à notre plateau tournant, nous avons pu faire l’acquisition de points de vue rapprochés et distribués autour de l’objet. Seul YCB-VIDÉO propose des points de vue plus rapprochés, car acquis à 30 images par seconde, mais ils ne sont pas distribués autour de l’objet. Ils suivent eux la trajectoire de la caméra. Et grâce à la caractéristique amovible de notre plateau tournant, nous avons pu photographier les arrière-plans ce qui est une première.

Chapitre 14

Résultats de détection 3D

Ce dernier chapitre présente les résultats de PoseNet et BB8 sur notre jeu de données. Notre jeu de données nous permet de travailler sur deux axes nouveaux.

Premièrement, notre jeu de données à des points de vue pris tout les 0.5° . Il nous permet donc de vérifier si ces deux méthodes sont capables de meilleurs résultats à 5° et 5 cm que ceux enregistrés sur les autres jeux de données, et si les résultats avec des seuils plus bas sont meilleurs. Par exemple à 1° et 1 cm voir même 0.5° et 0.5 cm . Ainsi nous saurons si les performances de ces méthodes sont limitées par leur conception ou par les données utilisés pour les entraîner. Améliorer la précision des méthodes, ou trouver des méthodes plus précises, est essentiel dans le cas d'application en industrie, car une précision de 5° et 5 cm n'est pas suffisante.

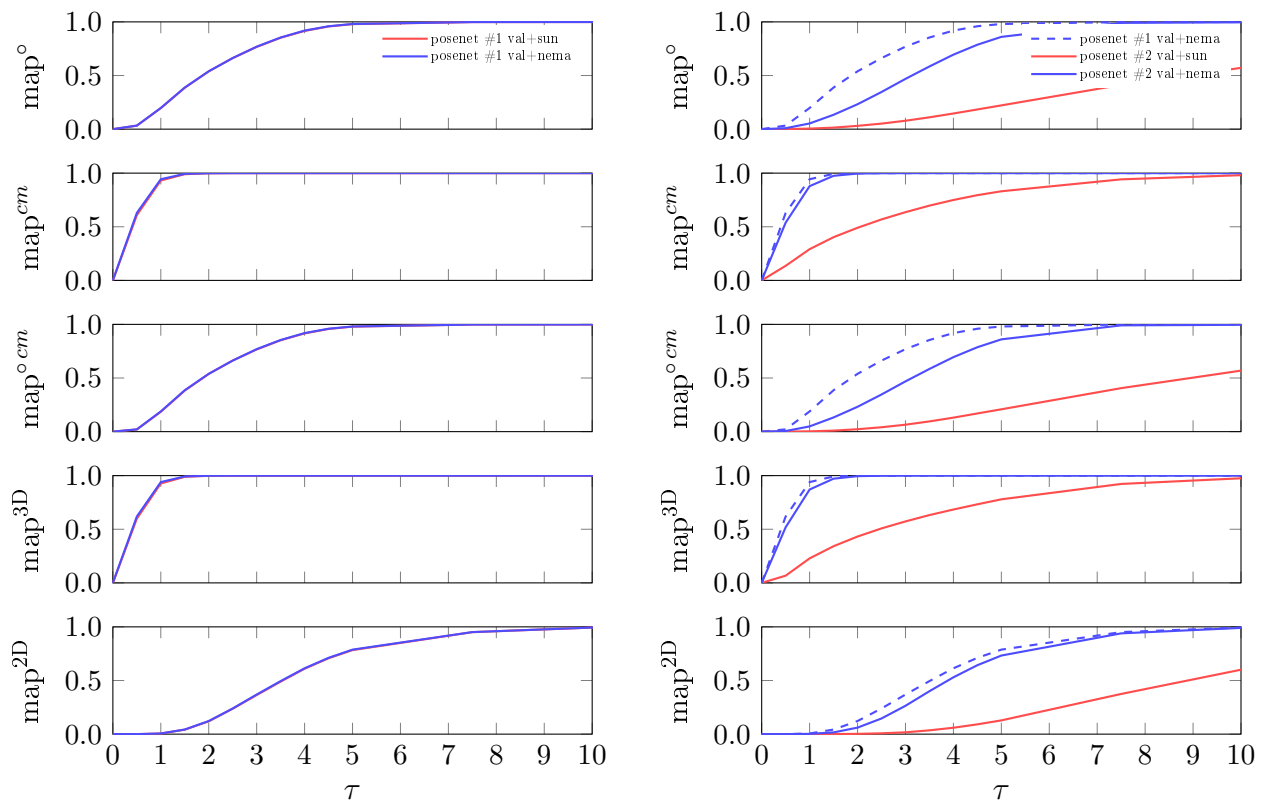
Deuxièmement, notre jeu de données est aussi le seul à fournir les arrières-plans réels. Nous avons la possibilité de les utiliser, ou de ne pas les utiliser. Dans le dernier cas, nous pouvons utiliser les images de SUN comme nous les avons utilisées dans nos autres expériences. Cela va nous permettre de savoir si les arrières-plans réel sont préférables pour l'entraînement de part les informations de contexte qu'ils apportent, ou si des arrières-plans quelconques sont préférables de part la variété des couleurs, des textures et des formes qu'ils proposent. Ainsi, nous pourrions valider le protocole utilisé par les méthodes de l'état de l'art. Nous pourrions aussi vérifier si une méthode entraînée avec un type d'arrières-plans généralise sur l'autre.

Ces expériences vont donc nous permettre de savoir s'il est possible d'améliorer la précision de PoseNet et BB8 grâce à des données d'entraînement plus complètes et de trouver le protocole d'entraînement qui apporte les meilleurs résultats.

Nous savons déjà que PoseNet et BB8 sont tout les deux capables de traiter de plusieurs objets grâce à nos expériences sur LINEMOD et YCB-VIDÉO. Notre implémentation TensorFlow nous contraint à exporter les photographies de toutes les instances d'objet individuellement dans un fichier "tfrecords", ce qui permet de charger les données plus rapidement et d'accélérer l'entraînement. Mais notre jeu de données comprend plus d'images que tous les jeux de données précédemment considérés. Sans modifier notre implémentation, le fichier "tfrecords" aurait dépasser les 2To , la taille de notre disque dur SSD. Nous avons donc considéré le moteur dans son ensemble comme un seul objet et c'est la pose de sa base qui est utilisée dans chaque image.

14.1 PoseNet

L'entraînement de PoseNet est tout d'abord réalisé suivant le même protocole que sur LINEMOD, T-LESS et YCB-VIDÉO. Les images sont recadrées à 227×227 pixels. Chaque



(1) Résultat de l'entraînement de PoseNet avec les avant-plans de NEMA et les arrière-plans de SUN.

(2) Résultat de l'entraînement de PoseNet avec les avant-plans de NEMA et les arrière-plans de NEMA.

FIGURE 14.1 – Résultats de PoseNet sur NEMA avec les arrière-plans remplacés par des images de SUN (1) et avec les arrière-plans réel (2).

image est centrée sur l'objet qui y figure. Grâce aux masques de segmentation leurs arrière-plans sont remplacés par des cadres pris aléatoirement dans dix images de SUN. Et enfin nous faisons varier aléatoirement la luminosité, le contraste, la teinte et la saturation. La descente de gradient est répétée pendant 100 époques avec un taux d'apprentissage fixe de 0.001. Cet entraînement est noté *posenet#1*. En suivant nous ré-entraînons PoseNet mais cette fois ci avec les arrière-plans réels. Le reste de la préparation des images et de l'optimisation des paramètres est effectué de manière identique. Cet entraînement est noté *posenet#2*. Nous avons testé ces deux versions PoseNet sur les images de NEMA avec les arrière-plans remplacés par des images de SUN, noté *val + sun*, et avec les arrière-plans réels, noté *val + nema*. Les résultats sont présentés à la figure 14.1.

L'entraînement *posenet#1*, présenté à gauche dans la figure 14.1, généralise très bien sur les images *val + sun* et *val + nema* car comme on peut le voir il n'y a aucun décrochage entre les courbes rouges et bleus. La précision à $5^\circ \wedge 5\text{ cm}$ est de 98.04 % ce qui est considérablement meilleur que les résultats obtenus sur LINEMOD 22.85 %, TLESS 0 % et YCB-VIDÉO 0.97 %. Encore plus intéressant PoseNet obtient 18.85 % de précision à $1^\circ \wedge 1\text{ cm}$, soit quasiment aussi bien que le résultat mentionné ci-dessus sur LINEMOD à $5^\circ \wedge 5\text{ cm}$. Les images de NEMA ont été prises à des points de vus similaires à ceux de LINEMOD mais nous disposons de plus d'images, avec des points de vus pris plus densément tous les 0.5° . Ce résultat montre donc qu'avec une même méthode de détection 3D, plus d'images d'entraînements permettent d'améliorer significativement la précision des poses estimées. L'entraînement *posenet#2*, présenté à droite, nous montre que PoseNet entraîné avec les arrière-plans réel est moins bon par rapport au *posenet#1* sur les images de *val + nema* et encore moins bon sur les images de *val + sun*. La précision à $5^\circ \wedge 5\text{ cm}$ est de 86.03 % sur *val + nema*, et 20.82 % sur *val + sun*.

Ces deux entraînements nous montrent qu'il est préférable d'entraîner PoseNet avec des arrière-plans aléatoires *train + sun*. Ces images d'entraînement plus variées vont permettre à PoseNet de généraliser sur des images dont il connaît les arrière-plans (*val + sun*) et sur des images avec des arrière-plans qui lui sont inconnus (*val + nema*) mais plus simples. Entraîné avec les arrière-plans de NEMA (*train + nema*), PoseNet se spécialise sur les images de NEMA. Cela se remarque par le fait que la précision sur les images *val + sun* chute fortement. Mais malheureusement cette spécialisation sur *train + nema* est telle que les résultats sur *val + nema* chutent eux aussi. PoseNet ne tire donc pas parti de la moindre variété des images de *train + nema*.

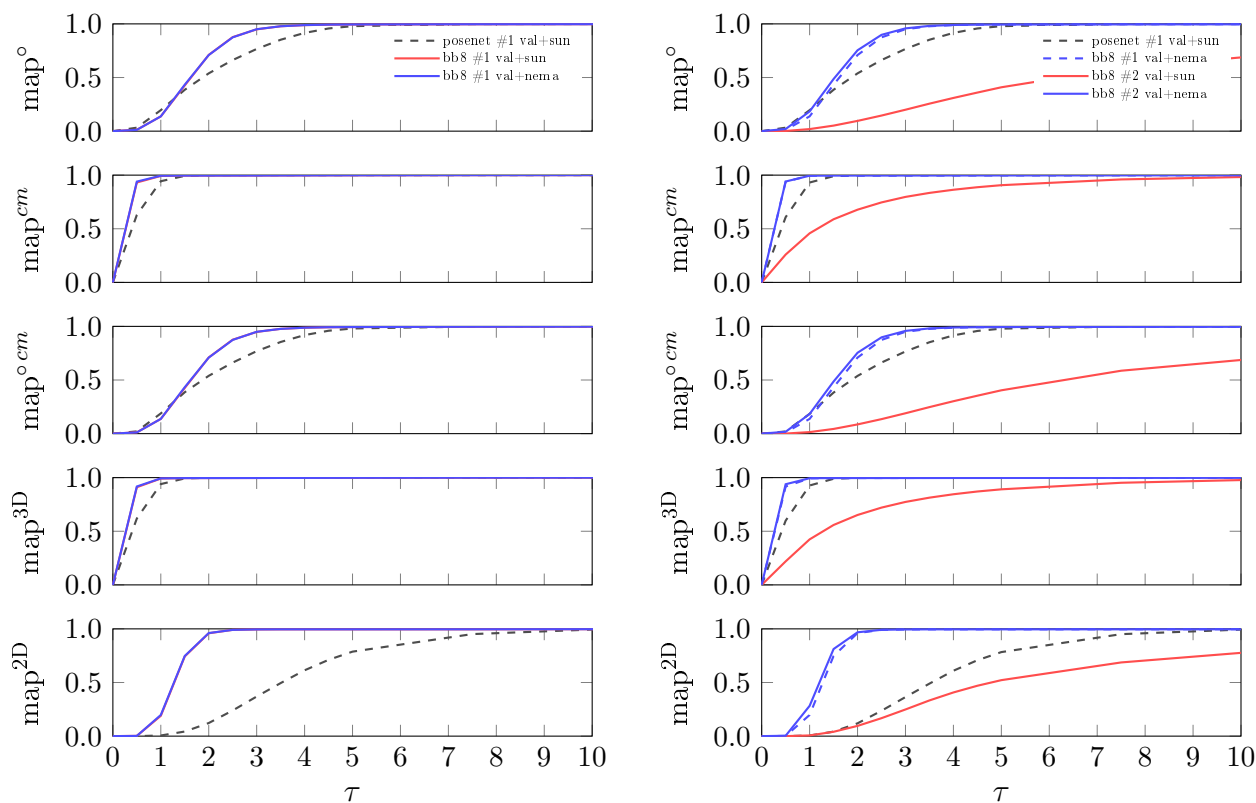
14.2 BB8

Pour BB8 nous procédons de la même manière. Nous l'entraînons une première fois sur les images *train + sun* et, une deuxième fois sur les images *train + nema*. Nous nommons ces deux entraînements *bb8#1* et *bb8#2*.

Les résultats de ces deux entraînements sont présentés à la figure 14.2. La partie de gauche présente les résultats de *bb8#1* comparés à ceux de *posenet#1*, et la partie de droite ceux de *bb8#2* comparés à *posenet#1* et *bb8#1*.

Les résultats du premier entraînement nous confirment ce que nous avons appris avec les jeux de données de l'état de l'art. BB8 est bien meilleur que PoseNet comme le montre toute les métriques avec des aires sous les courbes bien plus importantes pour les courbes bleus (*bb8#1*) par rapport aux courbes en pointillés noires (*posenet#1*). On remarque juste que PoseNet semble être meilleur pour estimer la rotation pour des seuils de précisions $\tau < 1^\circ$. Ce qui donne aussi l'avantage à PoseNet pour l'estimation de la rotation et de la translation au même seuil de précision. Mais la tendance est inverse pour des seuils plus tolérants.

Le deuxième entraînement nous montre que réaliser l'entraînement de BB8 sur les images



(1) Résultat de l'entraînement de BB8 avec les avant-plans de NEMA et les arrière-plans de SUN.

(2) Résultat de l'entraînement de BB8 avec les avant-plans de NEMA et les arrière-plans de NEMA.

FIGURE 14.2 – Résultats de BB8 sur NEMA avec les arrière-plans remplacés par des images de SUN (1) et avec les arrière-plans réel (2).

	LINEMOD		T-LESS		YCB-VIDÉO		NEMA	
	train	val	train	test	train	test	train+sun	val+nema
5°	68.38 %	28.49 %	96.31 %	3.31 %	88.18 %	3.99 %	97.99 %	98.04 %
5 <i>cm</i>	56.68 %	28.80 %	100.00 %	0.65 %	88.09 %	5.02 %	99.99 %	99.99 %
5° 5 <i>cm</i>	41.98 %	11.76 %	96.31 %	0.03 %	79.85 %	0.45 %	97.99 %	98.04 %
3D 5 <i>cm</i>	56.43 %	27.95 %	99.99 %	0.20 %	87.97 %	3.02 %	99.99 %	99.99 %
2D 5 <i>px</i>	02.42 %	0.59 %	40.95 %	0 %	2.39 %	0.01 %	78.51 %	78.77 %

TABLE 14.1 – Résultats complets de PoseNet.

	LINEMOD		T-LESS		YCB-VIDÉO		NEMA	
	train	val	train	test	train	test	train+nema	val+nema
5°	39.54 %	27.49 %	85.44 %	0.12 %	87.33 %	2.12 %	99.31 %	99.39 %
5 <i>cm</i>	84.19 %	72.05 %	97.80 %	0.13 %	91.65 %	15.49 %	99.75 %	99.69 %
5° 5 <i>cm</i>	35.50 %	22.85 %	84.05 %	0 %	82.27 %	0.97 %	99.28 %	99.38 %
3D 5 <i>cm</i>	83.73 %	71.20 %	97.78 %	0.11 %	91.60 %	13.65 %	99.75 %	99.69 %
2D 5 <i>px</i>	26.29 %	20.91 %	0.03 %	0 %	34.40 %	0.79 %	99.57 %	99.56 %

TABLE 14.2 – Résultats complets de BB8.

train+nema conduit à une spécialisation sur le jeu de donnée NEMA. Comme pour PoseNet on peut voir que les résultats de BB8 sur *val + sun* sont largement inférieurs à ceux sur *val+nema*. Par exemple, pour un seuil de $\tau = 1$ l'ADD 2D de *bb8#2* sur les images *val + sun* est de 0.22 % alors que sur *val + nema* elle est de 28.24 %. Mais contrairement à PoseNet, BB8 tire bien parti de cette spécialisation pour améliorer ses résultats sur *val+nema*, comme on peut le voir en observant les courbes bleues (*bb8#2val + nema*) par rapport aux courbes en tirets bleues (*bb8#1val + nema*). Par exemple, pour un seuil de $\tau = 1$ l'ADD 2D de *bb8#2* sur les mêmes images est de 28.24 % alors que pour *bb8#1* sur *val + nema* elle était de 19.73 %. Entraîner BB8 sur les arrière-plans réels a donc un intérêt, mais mitigé, car bien que cela permette de gagner presque 10 % de précision ADD 2D, ce gain se fait au détriment de la capacité de généralisation de BB8.

14.3 Conclusion des expériences

La constitution du jeu de données NEMA et la tenue de ces deux expériences avaient pour but de répondre principalement aux deux questions suivantes :

- Est-ce que la précision des méthodes d'estimations des 6 degrés de liberté peut être améliorée grâce à plus d'images d'entraînements couvrant plus de points de vus ?
- Est-ce que l'utilisation des arrière-plans réels permet d'augmenter la précision des mêmes méthodes ?

Si on reprend les tableaux 9.5 et 9.6 qui présentaient les résultats de PoseNet et BB8 sur les jeux de données de l'état de l'art, et qu'on y ajoute les résultats sur NEMA alors, la réponse à la première question est oui. Ces deux nouveaux tableaux sont visibles ci-dessus (14.1, 14.2). Par rapport aux résultats obtenus sur les jeux de données de l'état de l'art, les précisions de PoseNet et de BB8 ont toutes deux augmentées considérablement. La mesure de recalage 2D dans l'image, qui nous intéresse particulièrement dans le cadre de la réalité augmentée, atteint 78.77 % à 5 *px* sur NEMA (*val+nema*) pour PoseNet, alors qu'anciennement le meilleur résultat était de 0.59 % sur LINEMOD (*val*). BB8 obtient 99.56 % sur NEMA (*val+nema*), alors que précédemment son meilleur résultat était de 20.91 % sur LI-

NEMOD (val). On peut donc même dire que le manque d'image avec des étiquettes précises était la difficulté la plus importante à laquelle les méthodes considérées devaient faire face.

Pour ce qui est de l'utilisation des arrière-plans nous avons obtenu deux résultats différents. PoseNet, qui estime directement des paramètres en 3D (translation et rotation), semble réussir moins bien avec les arrière-plans réels. Et BB8, qui estime lui des points en 2D sur l'image qui sont ensuite utilisés par PnP pour retrouver les paramètres 3D, semble mieux réussir avec les arrière-plans réels. Les arrière-plans réels peuvent donc être bénéfiques comme néfastes, et il faut en tenir compte.

Dans les deux cas, c'est le même mécanisme qui est à l'œuvre : la sur-spécialisation. Entraînés avec les arrière-plan réel PoseNet et BB8 se sur-spécialisent tous les deux sur des images avec ces arrière-plans seulement. Leurs résultats sur les mêmes images mais avec des arrière-plans pris dans des images de SUN sont donc moins bons. Cette sur-spécialisation est normale et attendue puisqu'il est vrai que nos arrière-plans sont moins variés : la table est toujours la même et elle est posée devant le même mur. La différence est que PoseNet se sur-spécialise aussi d'avantage sur les points de vue d'entraînements, et ses résultats à des points de vue inconnus chutent alors significativement. La mesure de recalage 2D à 5 *px* perd 5.5 % de précision. BB8 lui arrive à mieux généraliser sur des points de vue nouveaux et sa précision pour la même mesure, mais à 1 *px* près, augmente de 8.51 %. Ce résultat démontre encore l'avantage de BB8 : prédire des points en 2D plutôt que d'estimer directement les paramètres de poses en 3D rend la méthode moins sujette à la sur-spécialisation.

Conclusion

Chapitre 15

Rappel du contexte et de la problématique

De nos jours la 3D est présente à tous les niveaux d'une entreprise et tout au long de la vie des produits. Les objets sont conçus en 3D grâce à des logiciels de conception sur ordinateurs (CAO). Ils sont intégrés dans des maquettes 3D pour en valider les dimensions et l'apparence. Ces maquettes virtuelles sont aussi utilisées pour faire la promotion numérique des produits. Les machines à commande numérique qui les produisent, utilisent le modèle 3D traduit en chemin d'outils, qu'elles soient soustractives ou additives. Les modèles 3D sont aussi utilisés pour illustrer les manuels d'assemblage, d'utilisation et de maintenance.

Une usine et les lignes de production qu'elle regroupe sont aussi des produits achetés par l'industriel. ITECA propose une solution de jumeau numérique appelé SmartUpp qui permet justement de créer des maquettes à l'échelle de sites industriels complets. Ces maquettes peuvent aussi être connectées à l'usine réel pour remonter les données, y compris de mouvements, et animer la maquette en temps réel. WITTY, qu'en ta lui, est une solution d'aide à la maintenance. Ce logiciel aide les techniciens à diagnostiquer des pannes en leurs demandant des valeurs de capteurs ou des observations sur la machine. Quand la 3D y a été intégré, il est devenu possible de montrer virtuellement au technicien quoi regarder, où intervenir, et, grâce aux animations, comment le faire.

Notre expérience nous montre que la 3D permet de créer des outils logiciels plus compréhensibles et plus interactifs. De ce fait les logiciels sont plus simple d'utilisation et décharge cognitivement les utilisateurs qui peuvent alors se concentrer plus sur leurs tâches réels. Dans la pratique, on observe donc des gains de temps importants et une plus forte appétence à utiliser les outils.

Ces outils 3D sont une avancée importante comparés aux documents numériques (PDF) ou papiers qui sont encore utilisés de nos jours par des grands groupes. Mais pour qu'elle soit adoptée communément, la 3D doit résoudre un problème qui lui est, pour le moment, inhérent : elle est affichée sur des supports 2D, ce qui pose plusieurs problèmes.

Que ce soit imprimé sur papier ou affiché sur des écrans, mêmes panoramiques ou dans des caves d'écrans, l'information 3D se retrouve toujours projetée en 2D sur son support d'affichage. Il existe donc toujours un effort cognitif d'interprétation, et donc des erreurs d'interprétations qui sont possibles. Ce problème s'accroît dans le cas de techniciens mobiles. Ils doivent porter avec eux le PC portable, la tablette ou le smartphone, qui occupe une de leurs mains, voir les deux. L'interaction avec ces supports est parfois rendue difficile par les équipements de protection, mais si ces équipements sont retirés, on fait alors face à

des problèmes de sécurité. Le simple fait de regarder un écran dans un milieu industriel est d'ailleurs une distraction qui pose des problèmes de sécurité. Pour outrepasser ces problèmes la 3D doit changer de support d'affichage ce qui pose aussi des problèmes de mise en œuvre.

L'affichage holographique a longtemps fait partie du rêve littéraire ou cinématographique. Aujourd'hui, on commence pourtant à s'en rapprocher, notamment avec les casques de réalité mixte, comme les HoloLens 2 de Microsoft. Ces casques permettent d'afficher la 3D sur deux écrans transparents placés devant les yeux de l'utilisateur. L'image s'affiche donc en perspective par dessus l'environnement réel et l'immersion des visuels virtuels dans le réel est totale.

Ces casques résolvent le problème matériel de l'affichage mais ils imposent aussi des contraintes de développement logiciel. En effet, les casques sont capables de se positionner par rapport à l'environnement global (une pièce, ses murs, etc...) et donnent donc un référentiel de scène. Mais il appartient au développeur de correctement placer ses visuels dans ce référentiel, pour que ces visuels se retrouvent correctement affichés en 3D dans le réel. Les casques fournissent des outils pour détecter les surfaces planes comme le sol et les tables, ou encore détecter des objets du quotidien comme des chaises. Ce qui permet de positionner des visuels à ces endroits mais, cela est surtout utile pour le développement de jeux. Dans notre cas, nous avons besoin de retrouver les positions d'objets réels que nous allons ensuite augmenter avec des visuels. Il nous appartient donc de développer cette brique logicielle. Pour ce faire, nous avons accès aux capteurs du casque comme des caméras RGB ou en niveau de gris, des caméras de profondeurs et les données de la centrale inertielle. Mais nous devons aussi respecter des contraintes de ressources sur les casques comme une mémoire et une puissance de calcul limités. Grâce à la caméra du casque, nous espérons pouvoir estimer les six degrés de liberté des objets à augmenter.

Chapitre 16

Résultats des expériences

L'état de l'art nous a montré que ce problème logiciel pouvait se résoudre avec l'aide de réseaux de neurones convolutionnels. Plusieurs architectures de réseaux et plusieurs façons de modéliser la pose existent. Le choix de l'architecture est le premier à prendre en considération. Il faut choisir un réseaux dont l'architecture donne des résultats précis et qui puisse être déployé sur le casque qui a des ressources limitées (RAM, CPU). Dans notre cas, les réseaux dit "mobiles" sont une option sérieuse (GoogLeNet, MobileNet, Xception). Le deuxième point dont il faut tenir compte, est la façon de modéliser la pose prédite. On peut soit prédire directement les paramètres 3D recherchés, soit prédire d'abord des paramètres 2D sur l'image observée et utiliser PnP avec les modèles 3D pour retrouver ensuite les paramètres 3D.

Pour ce qui est du premier point, nos expériences ont montré que les méthodes mobiles sont aussi, voire plus performantes que des réseaux de neurones convolutionnels classiques comme VGG quand il s'agit de reconnaître les objets (classification). Pour cette tâche de classification, nous avons testé AlexNet, VGG 19 et GoogLeNet sur LINEMOD, T-LESS et YCB-VIDÉO. Sur LINEMOD et YCB-VIDÉO les trois réseaux donnent d'excellents résultats mais c'est GoogLeNet qui arrive en tête avec 99.91 % et 93.17 % de précision top-1 sur les images de tests. Sur T-LESS seul AlexNet donne des résultats mais qui restent limités : 47.72 % de précision top-1. D'après nous, ces mauvais résultats sur T-LESS sont dus aux objets qui partagent des sous-parties sans que cela ne soit traduit dans les étiquettes. Ainsi selon les points de vus, les objets ne peuvent être différenciés. Pour conclure sur le choix de l'architecture, on peut dire que les architectures mobiles comprennent moins de paramètres et sont tout-à-fait suffisantes pour les jeux de données que nous utilisons qui présentent moins de catégories d'objets. Pour la tâche d'estimation des six degrés de liberté nous avons testé PoseNet et BB8. PoseNet est construit avec l'architecture de GoogLeNet et BB8 avec celle de VGG. Mais comme nous allons le voir au deuxième point ces méthodes se différencient surtout par leur façon de modéliser la pose.

Pour ce qui est du deuxième point, nos expériences sur jeux de données de l'état de l'art nous ont laissé perplexe. Nous voulions pouvoir comparer objectivement PoseNet et BB8. C'est pourquoi nous avons souhaité calculer les mêmes mesures de précision sur les mêmes images de test, après avoir entraîné les réseaux sur les mêmes images. Nous n'avons pas souhaité utiliser de rendus 3D et PoseNet ne nous permet pas d'utiliser d'augmentations d'images qui déformeraient les objets. En pratique nous pouvons seulement recadrer, redimensionner et faire varier les couleurs dans les images.

Dans ces conditions PoseNet et BB8 donnent de mauvais résultats. Sur LINEMOD Po-

seNet obtient 0.59 % de précision de recalage 2D et BB8 20.91 %. BB8 avec plus d'augmentation d'images, et le module supplémentaire de correction des prédictions, est capable de biens meilleurs résultats. Mais les images d'entraînement utilisées dans ce cas ne seraient pas utilisables pour PoseNet. Sur T-LESS et YCB-VIDÉO, PoseNet et BB8 ne donnent aucun résultat sur les images de test car ils se sur-spécialisent sur les images d'entraînements.

À ce stade, il ne nous était pas possible de dire si ces mauvais résultats étaient dus à notre utilisation des jeux de données ou aux méthodes utilisées. Mais nous avons plus de réserves sur les jeux de données et notre protocole de préparation des images : Tout d'abord, LINEMOD et T-LESS ne sont pas conçus pour entraîner des réseaux de neurones. LINEMOD ne propose pas d'images d'entraînement et les points de vue limités (~ 1200 /objet) doivent donc être répartis entre entraînement et validation. T-LESS propose, lui, des images d'entraînements, mais peu, et trop différentes des images de test. YCB-VIDÉO est le premier jeu de données à destination des réseaux de neurones. Mais les images d'entraînement sont tirées de séquences vidéo différentes de celles de tests.

Les résultats de PoseNet et BB8 ne peuvent pas être améliorés sur ces jeux de données sans utiliser des rendus 3D pour couvrir les points de vue absents des images d'entraînements. Les résultats de BB8 pourraient être améliorés dans bien des cas grâce à des augmentations d'images qui déformeraient les objets pour créer les points de vue inconnus (translations, rotations et transvections). Mais ces transformations sont incompatibles avec les étiquettes de PoseNet. Il est donc difficile de faire généraliser BB8 et PoseNet sur les jeux de données de l'état de l'art.

C'est pourquoi nous avons réalisé notre propre jeu de données avec pour objectif principal d'acquérir un nombre très important d'images couvrant des points de vue variés et denses (proche les uns des autres). Cet objectif est double. Premièrement, il nous faut beaucoup d'images pour entraîner un réseau de neurones. Deuxièmement, nous souhaitons pouvoir augmenter la précision des méthodes considérées dont les mesures étaient jusqu'à présent calculées à 5 *cm* et 5°. Par "dense" nous voulons dire que nous avons reculé notre caméra centimètre par centimètre de l'objet, et que le plateau tournait par pas de 0.5°. Nous avons donc des points de vue extrêmement précis et proches les uns des autres, qui vont nous permettre de calculer des mesures de précisions à 1 *cm* et 1°, voire moins. Pour assurer la précision de ces calculs, il était essentiel que nos étiquettes soient d'excellente qualité, ce que nous avons réussi à produire grâce à notre plateau tournant et notre protocole d'acquisition. Au final, nous avons photographié 172 800 images avec 691 200 instances d'objets. Tout cela avec une précision de recalage des étiquettes en moyenne de 1.0899 *mm* dans les cartes de profondeur avec seulement 10.37 % de pixels ignorés. Ce résultat dépasse les précisions obtenues dans les images de LINEMOD (1.49 *mm*/7.30 %), TLESS (train 5.36 *mm*/5.46 %, test 3.02 *mm*/7.74 %) et YCB-VIDÉO (train 1.11 *mm*/20.60 %, test 0.98 *mm*/19.38 %)

Nous avons par la suite ré-évalué PoseNet et BB8 sur notre jeu de données. Avec nos images les deux méthodes fonctionnent et donnent de bons résultats : PoseNet obtient 78.77 % de précision de recalage 2D à 5 *px* et BB8 obtient 99.56 %. Face aux mêmes images BB8 obtient des résultats significativement meilleurs (+20.79 %). BB8 sous-traite le problème de perspective inverse à PnP, ce qui permet au réseau de répondre à un problème formulé dans le plan image. Cela semble être une meilleure stratégie comparé à PoseNet qui traite un problème formulé dans le repère 3D de la caméra.

Grâce à notre jeu de données nous avons des points de vue plus denses. Nous avons donc pu abaisser notre seuil de tolérance τ dans les mesures de précisions qui à $\tau = 5$ sont

quasiment toutes à leur maximum. Avec $\tau = 1$ nous avons mesuré que PoseNet et BB8 ont tous les deux des précisions de recalage 3D supérieures à 99 %. Nos expériences montrent que c'est la rotation qui est la plus difficile à estimer. PoseNet a du mal à prédire des rotations au delà d'une précision de l'ordre de 4° . BB8 permet d'obtenir des scores comparables à une précision de l'ordre de 2° . Enfin nous avons montré que BB8 dépasse largement PoseNet sur la mesure de recalage 2D, qui, on le rappelle, est la plus importante en réalité augmentée. Dès que nous abaissons la valeur de τ la précision de recalage 2D de PoseNet chute. À $4px$ elle passe déjà sous les 50 %. Pour BB8 c'est complètement différent, la précision de recalage 2D se maintient à des valeurs excellentes jusqu'à $2px$. À $1.5px$ elle toujours très correcte avec plus de 75 % d'objets correctement recalés. Et c'est seulement à partir de $1px$ qu'elle passe sous les 50 %. BB8 domine complètement les résultats de précisions de recalage 2D.

BB8 est meilleur en recalage 2D car la fonction de coût utilisée pour l'entraîner minimise justement l'erreur de recalage 2D. Ces résultats montrent aussi que de bons résultats de recalage 2D permettent d'obtenir de bons résultats de recalage 3D grâce à E-PnP. Mais qu'inversement, il est plus difficile d'obtenir de bons résultats de recalage 2D avec un modèle apprenant des paramètres en 3D. Cela est dû, premièrement, au fait qu'une petite erreur d'estimation de la rotation 3D peut entraîner de gros écarts en 2D. Et deuxièmement, au fait qu'apprendre à prédire la position de points 2D à partir d'images est plus simple car les points 2D sont dans le même repère que la donnée d'entrée.

Nos expériences avec BB8 nous ont aussi montré que quand on dispose de peu d'images d'apprentissage (LINEMOD, T-LESS) E-PnP permet de corriger certaines erreurs. Dans ce cas, E-PnP apporte une meilleure solution au problème de projection inverse qu'un réseau qui apprendrait aussi cette étape (PoseNet). En moyenne la précision de recalage à $5px$ est de 1 à 2 % après utilisation de PnP. Ce gain diminue quand plus de points de vue sont présentés dans les images d'entraînement, comme c'est le cas avec notre jeu de données NEMA. Dans ce cas, on pourrait utiliser un algorithme plus simple pour PnP et économiser des ressources de calculs.

Chapitre 17

Contributions

Les contributions de cette thèse sont triples :

Premièrement, nous avons réalisé une rigoureuse comparaison entre AlexNet, GoogLeNet et VGG pour la classification d'images sur les jeux de données d'estimation des six degrés de liberté. Cette première réalisation nous a permis de valider l'implémentation des réseaux que nous utiliserons par la suite pour implémenter les réseaux d'estimation des six degrés de liberté. Elle nous a aussi permis de comparer plusieurs techniques de préparation d'images et leurs impacts sur les résultats de classification. Certaines des techniques que nous avons testées peuvent aussi être utilisées pour l'entraînement des PoseNet et BB8, ou pré-entraîner les ossatures de ces derniers.

Deuxièmement, nous avons comparé PoseNet et BB8 pour l'estimation des six degrés de liberté sur les mêmes jeux de données que précédemment, et avec la même préparation d'images. Cette comparaison mettant en place les mêmes images d'entraînement avait pour but de permettre une comparaison objective des architectures et des modélisations de la pose de ces deux méthodes (VGG contre GoogLeNet, 3D contre 2D). Au final ces tests ont surtout démontré que les jeux de données de l'état de l'art ne sont pas adaptés à l'entraînement de réseaux de neurones, sans y ajouter des rendus 3D, ce que nous ne souhaitons pas utiliser.

Lors de ces tests nous avons néanmoins observé des cas de sur-spécialisations qui mettent en avant certains aspects intéressants des méthodes considérées. Nous avons montré, par exemple, que si PoseNet est entraîné avec trop peu d'images d'entraînement, les deux termes de la fonction de coût entrent en compétition et déstabilisent l'entraînement. Cette observation a fait l'objet d'une publication à VISAPP 2019 :

Limitations of Metric Loss for the Estimation of Joint Translation and Rotation

Philippe Pérez de San Roman, Pascal Desbarats, Jean-Philippe Domenger, Axel Buendia
International Joint Conférence on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)
25-27 Février 2019
Prague, République Tchèque
<https://www.scitepress.org/Link.aspx?doi=10.5220/0007525005900597>

N'ayant pas pu comparer PoseNet et BB8 comme nous le souhaitions, et conscient que le problème tenait principalement aux jeux de données utilisés, nous avons réalisé un nouveau jeu de données. La réalisation de ce jeu de données et sa comparaison qualitative face à LINEMOD, T-LESS et YCB-VIDÉO a fait l'objet d'une publication à VISAPP 2022 :

NEMA : 6-DoF Pose Estimation Dataset for Deep Learning

Philippe Pérez de San Roman, Pascal Desbarats, Jean-Philippe Domenger, Axel Buendia
International Joint Conférence on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP).
6-8 Février 2022

En ligne

<https://www.scitepress.org/Link.aspx?doi=10.5220/0010913200003124>

Ce nouveau jeu de données nous a permis de réaliser un comparaison plus complète que ce que nous aurions pu réaliser avec LINEMOD, T-LESS et YCB-VIDÉO, car nous avons maintenant accès aux arrière-plans réels. Cela nous a permis de comparer les résultats obtenus avec BB8 et PoseNet selon que l'on entraîne le réseau avec ou sans les arrière-plans réels. Nous avons démontré que l'utilisation des arrière-plans réels, ou d'images aléatoires de substitutions, a un impact sur les performances. Les arrière-plans aléatoires permettent d'éviter la spécialisation du réseau ce qui peut être bénéfique comme néfaste pour les performances selon le réseau de neurones utilisé.

Chapitre 18

travaux futurs

Premièrement, il est possible de poursuivre et d’approfondir les tests réalisés sur les jeux de données existants :

- Nous avons entraîné PoseNet et BB8 sur LINEMOD. Les performances de BB8 peuvent être encore améliorées grâce à plus d’augmentations d’images. Mais, comme nous l’avons mentionné, ces images ne seront pas utilisables pour PoseNet. De plus, LINEMOD OCCLUDED fournit une séquence de test avec plus d’occlusions. Nous pouvons donc évaluer les modèles déjà entraînés sur cette séquence. Les résultats seront sans doute moins bons que sur LINEMOD, surtout pour PoseNet.
- Nous avons vu que les objets de T-LESS sont étiquetés individuellement alors qu’ils partagent des sous-parties. Il est possible de séparer manuellement ces parties d’objets et de créer une arborescence de pièces à la façon de WordNet [80]. Par la suite toutes les étiquettes du jeu de données peuvent être modifiées automatiquement. Cela inclu les étiquettes de catégorisation, les masques de segmentation, vecteurs de translations, matrices de rotations, les boîtes et rectangles encadrants et toutes les étiquettes calculées à partir de ces derniers (YOLO, BB8...). Cette représentation des objets permettra d’améliorer les résultats de PoseNet et BB8. Mais T-LESS reste un jeu de données qui n’a pas été conçu pour l’apprentissage profond. Les résultats resteront donc limités si on n’utilise pas d’autres images d’entraînement : une partie des images de tests, des rendus 3D, ou des séquences nouvellement photographiées.
- Pour YCB-VIDÉO l’augmentation des images d’entraînement peut permettre à BB8 de ne pas se sur-spécialiser sur les images d’entraînement, et d’améliorer ses résultats sur les images de tests. Les résultats de PoseNet ne peuvent pas être améliorés sans utiliser des rendus 3D ou de nouvelles images.
- Pour toute ces expériences nous avons toujours effacé les arrière-plans en le masquant avec une image aléatoire. Varier la technique utilisée pour mélanger les images d’arrière et d’avant-plans peut impacter la précision des résultats [31]. Mais il nous faudrait implémenter dans *TensorFlow* la combinaison d’images avec un mixage Gaussienne ou de Poisson. Cela s’applique aussi à nos expériences sur NEMA.

Deuxièmement, notre jeu de données est le premier à proposer les arrière-plans réels avec des étiquettes de grande qualités pour entraîner et évaluer les méthodes d’estimation des six degrés de liberté basés sur les réseaux de neurones. Il ouvre donc de nombreuses possibilités.

- Dans nos expériences, nous avons seulement utilisé des réseaux de neurones convolutifs classiques. Mais ces derniers n’utilisent pas les informations présentes dans les arrière-plans puisqu’ils construisent des représentations centrées sur les objets. Or intuitivement nous savons que l’être humain est très bon à estimer les distances à des objets.

Même avec un seul œil, nous avons des à priori de tailles sur les objets et nous sommes capable de comparer des objets côte-à-côte, ou l'un derrière l'autre, pour en approximer la taille. D'autres types de couches de convolution existent sur ce principe, notamment les convolutions récurrentes [73]. Elles pourraient permettre d'améliorer les résultats de détection. Mais il faudrait aussi valider leur utilisation de ressources sur les casques.

- De nombreuses autres techniques d'estimation des six degrés de liberté sont apparues au cours de cette thèse. Nous en avons présentées plusieurs dans l'état de l'art. Certaines sont dérivées de PoseNet et BB8 mais d'autres utilisent des représentations nouvelles, comme les "embeddings". Nous aurions aimé avoir le temps de les tester mais il paraissait plus intéressant de produire d'abord un jeu de données de meilleure qualité.
- Enfin, nous n'avons utilisé que les images couleur dans nos expériences mais notre jeu de données et ceux de l'état de l'art proposent aussi des cartes de profondeur. Grâce aux arrière-plans réels, nous avons d'ailleurs des cartes de profondeur complètes, même au abords des objets. Toute une famille de méthodes d'estimation des six degrés de liberté utilisent les cartes de profondeur et il serait intéressant de les évaluer sur notre jeu de données avec et sans les arrière-plans réels comme nous l'avons fait pour les images couleur.

Cinquième partie

Annexes

Annexe A

Détails des entraînements réalisés

Introduction

Dans ce chapitre annexe nous apportons des détails sur les entraînements présentés à la partie III chapitre 9.

Communs

Les détails d'implémentations suivant sont communs à tous les entraînements sauf indications contraires.

↳ Descente de gradient :
⊞ Algorithme : Descente de gradient stochastique
% Taux d'apprentissage : 0.001

Pour la classification d'images, la fonction d'activation utilisée après la dernière couche dense est systématiquement le softmax. De même la fonction de coût utilisé est toujours la entropie croisée catégorique.

⊞ Modèle :
↳ Activation : Softmax
↳ Descente de gradient :
⊞ Fonction de coût : Entropie croisée catégorique

Pour l'estimation des six degrés de liberté, la fonction d'activation utilisée après la dernière couche dense est systématiquement la fonction identité.

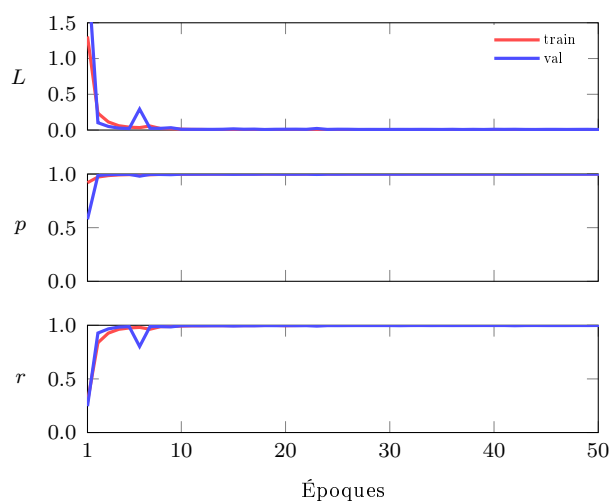
⊞ Modèle :
↳ Activation : Linéaire

A.1 VGG 19 / LINEMOD

Implémentation

- 🏠 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 15 bits
- 📁 Images :
 - 📄 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 📄 Préparation :
 - Recadrage à 224×224 px centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - Nombre d'époques : 1000

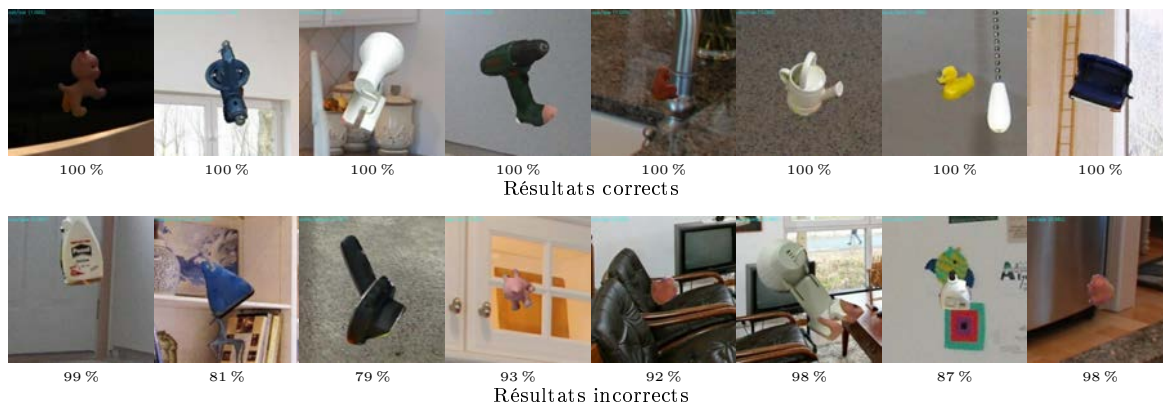
Entraînement



Scores

Mesure	τ	train	val
coût		$1.3e^{-7}$	0.0088
précision	.75	1.0	0.9992
rappel	.75	1.0	0.9988
vrais négatifs	.25	1.0	0.9998
vrais positifs	.75	1.0	0.9988
vrais	.75	1.0	0.9998
faux négatifs	.25	1.0	0.9991
faux positifs	.75	1.0	0.9999

Résultats



A.2 VGG 19 / TLESS / #1

Implémentation

📦 Modèle :

🔗 Réseau : VGG 19

📄 Entrée : $224 \times 224 \times 3$

📄 Sortie : vecteur logique 30 bits

📁 Images :

📄 Jeu de données : TLESS

↻ Répartition : 37584 entraînement et 69552 validation/test

🍷 Préparation :

Recadrage à 384×384 px centré sur l'objet

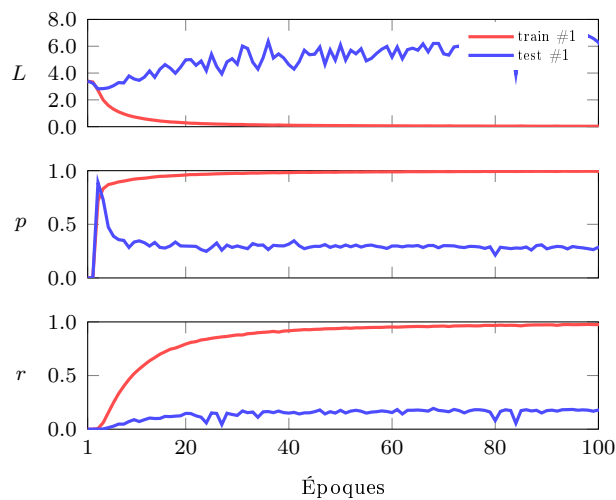
🔗 Mise à l'échelle à 224×224

📄 Remplacement de l'arrière-plan avec une image de SUN

📉 Descente de gradient :

Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût		0.0307	5.7139
précision	.75	0.9946	0.2858
rappel	.75	0.9757	0.1622
vrais négatifs	.25	0.9991	0.9684
vrais positifs	.75	0.9757	0.1622
vrais	.75	0.9990	0.9585
faux négatifs	.25	0.9940	0.2446
faux positifs	.75	0.9998	0.9860

Interprétation

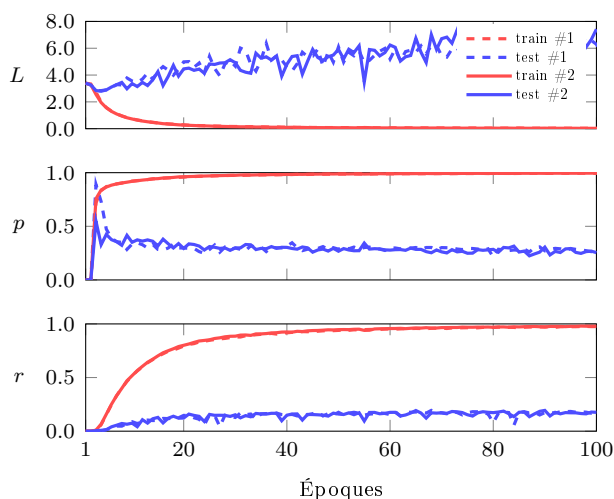
VGG 19 apprend bien à reconnaître les objets de TLESS dans les images d'entraînements. Sur les images de test il est capable de correctement identifier les objets absents mais produit beaucoup de faux négatifs. Sans augmentations d'images VGG ne généralise donc pas bien sur les images de test.

A.3 VGG 19 / TLESS / #2

Implémentation

- 🏠 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 30 bits
- 📁 Images :
 - 📄 Jeu de données : TLESS
 - 🔄 Répartition : 37584 entraînement et 69552 validation/tests
 - 🍹 Préparation :
 - 📄 Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - Recadrage à 384×384 px centré sur l'objet
 - 🔗 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0265	7.4218
précision	.75	0.9955	0.2545
rappel	.75	0.9955	0.1758
vrais négatifs	.25	0.9786	0.9679
vrais positifs	.75	0.9992	0.1758
vrais	.75	0.9991	0.9554
faux négatifs	.25	0.9950	0.2476
faux positifs	.75	0.9998	0.9822

Interprétation

Cet entraînement montre que les exemples largement ou complètement occultés ont un impact négatif sur le rappel. Ce qui est prévisible, car il vont compter comme des faux négatifs, alors qu'il ne sont pas réellement positifs. Leur présence dans les images de tests n'est pas justifiée. Les éliminer va permettre de travailler sur les autres challenges du jeu de données. Cet entraînement a pour but de servir de référence pour les suivants.

A.4 VGG 19 / TLESS / #3

Implémentation

📦 Modèle :

- 🔗 Réseau : VGG 19
- 📄 Entrée : $224 \times 224 \times 3$
- 📄 Sortie : vecteur logique 30 bits

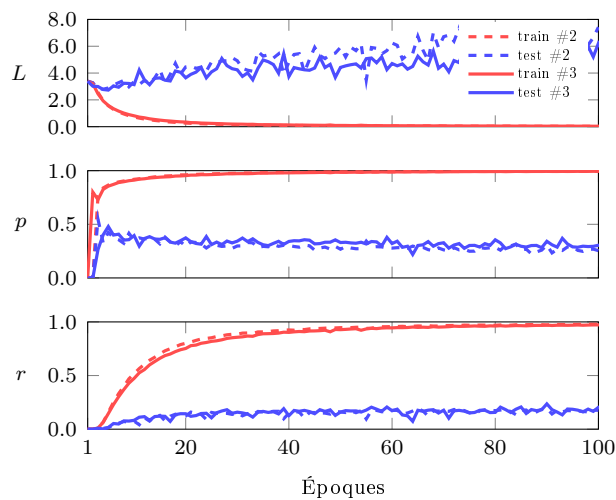
📁 Images :

- 📄 Jeu de données : TLESS
- 🔄 Répartition : 37584 entraînement et 69552 validation/tests
- 🍹 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - 📐 Recadrage centré sur l'objet à $384 \times 384 px$, fois un facteur aléatoire entre 0.95 et 1.05.
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN

📉 Descente de gradient :

- Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0372	5.7055
précision	.75	0.9937	0.3016
rappel	.75	0.9710	0.1857
vrais négatifs	.25	0.9990	0.9683
vrais positifs	.75	0.9710	0.1857
vrais	.75	0.9988	0.9585
faux négatifs	.25	0.9930	0.2787
faux positifs	.75	0.9997	0.9851

Interprétation

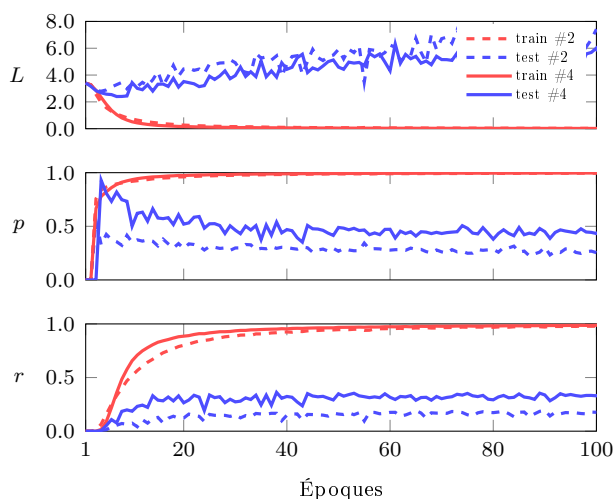
Amélioration de la précision de 5 % et du rappel de 1 %. Les changements aléatoire d'échelles sont donc une bonne solution pour adresser la différence d'échelles des objets entre les images d'entraînement et de test.

A.5 VGG 19 / TLESS / #4

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 30 bits
- 📁 Images :
 - 📄 Jeu de données : TLESS
 - 🔄 Répartition : 37584 entraînement et 69552 validation/tests
 - 🍷 Préparation :
 - 📉 Fitrage des objets occultés à plus de 70% ce qui produit 62681 images de tests
 - 📏 Recadrage aux dimensions du rectangle encadrant plus une marge de $16px$
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0175	5.2904
précision	.75	0.9968	0.4650
rappel	.75	0.9867	0.3494
vrais négatifs	.25	0.9995	0.9749
vrais positifs	.75	0.9867	0.3494
vrais	.75	0.9994	0.9649
faux négatifs	.25	0.9966	0.4331
faux positifs	.75	0.9998	0.9861

Interprétation

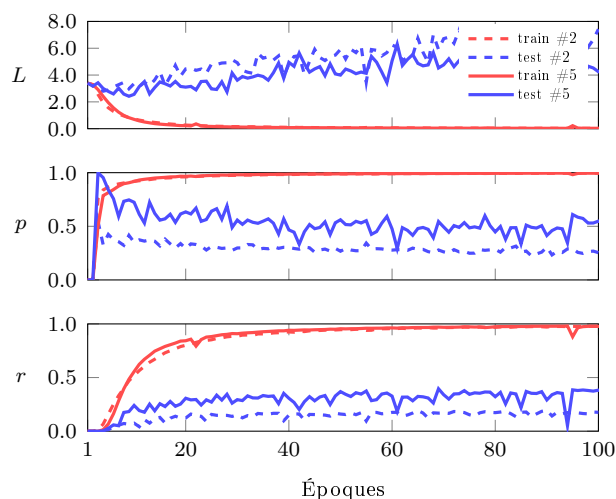
Amélioration significative de la précision et du rappel de 16% tout les deux. Mais les rectangles encadrants ne sont pas sensés être utilisés pour la classification d'images. Ce protocole n'est cohérent que pour affiner l'entraînement de VGG sur TLESS afin de l'utiliser comme ossature pour la détection 2D ou 3D.

A.6 VGG 19 / TLESS / #5

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 30 bits
- 📁 Images :
 - 📄 Jeu de données : TLESS
 - 🔄 Répartition : 37584 entraînement et 69552 validation/tests
 - 🍹 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - 📐 Recadrage centré sur l'objet à $384 \times 384 px$ plus une marge de $16 px$, fois un facteur aléatoire entre 0.95 et 1.05.
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - ☑ Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0294	4.3678
précision	.75	0.9946	0.5330
rappel	.75	0.9774	0.3682
vrais négatifs	.25	0.9992	0.9752
vrais positifs	.75	0.9774	0.3682
vrais	.75	0.9990	0.9681
faux négatifs	.25	0.9942	0.4581
faux positifs	.75	0.9998	0.9888

Interprétation

Amélioration de 6 % de la précision et de 2 % du rappel. Nouvelle confirmation de l'importance du facteur d'échelle aléatoire pour améliorer la généralisation sur les images de test.

A.7 VGG 19 / TLESS / #6

Implémentation

📦 Modèle :

🔗 Réseau : VGG 19

📄 Entrée : $224 \times 224 \times 3$

📄 Sortie : vecteur logique 30 bits

📁 Images :

📄 Jeu de données : TLESS

🔄 Répartition : 37584 entraînement et 69552 validation/tests

🍹 Préparation :

📉 Filtration des objets occultés à plus de 70% ce qui produit 62681 images de tests

Recadrage à 384×384 px centré sur l'objet

🔗 Mise à l'échelle à 224×224

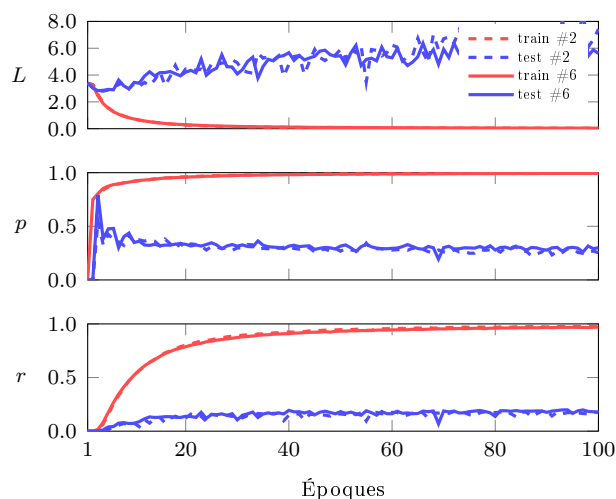
😊 Remplacement de l'arrière-plan avec une image de SUN

💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement

📉 Descente de gradient :

Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0372	5.6082
précision	.75	0.9941	0.2998
rappel	.75	0.9705	0.1765
vrais négatifs	.25	0.9990	0.9686
vrais positifs	.75	0.9705	0.1765
vrais	.75	0.9988	0.9588
faux négatifs	.25	0.9936	0.2646
faux positifs	.75	0.9998	0.9857

Interprétation

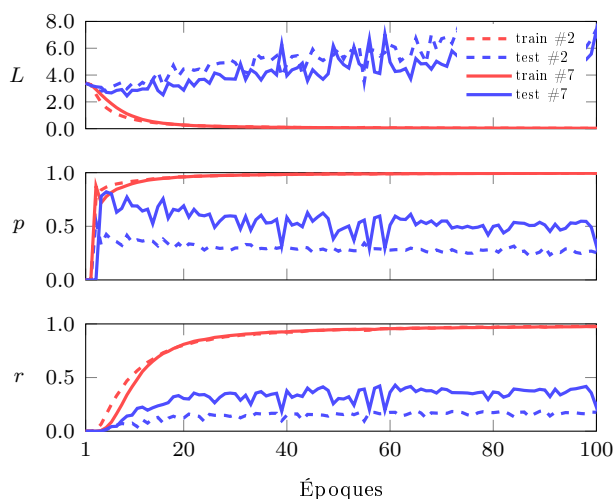
Par rapport à l'entraînement #2 la précision a progressé de 4% sans impact sur le rappel. Les variations aléatoires de luminosité, contraste, teinte et saturation aident donc bien VGG à généraliser sur les images de test.

A.8 VGG 19 / TLESS / #7

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 30 bits
- 📁 Images :
 - 📄 Jeu de données : TLESS
 - 🔄 Répartition : 37584 entraînement et 69552 validation/tests
 - 🍹 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - 📐 Recadrage centré sur l'objet à $384 \times 384 px$ plus une marge de $16 px$, fois un facteur aléatoire entre 0.95 et 1.05.
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 🌟 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0336	5.3677
précision	.75	0.9939	0.5112
rappel	.75	0.9733	0.3790
vrais négatifs	.25	0.9991	0.9753
vrais positifs	.75	0.9733	0.3790
vrais	.75	0.9989	0.9672
faux négatifs	.25	0.9935	0.4466
faux positifs	.75	0.9997	0.9875

Interprétation

Pour cet entraînement, nous avons combiné toutes les préparations d'images ayant données les meilleurs résultats. Les résultats obtenus sont supérieurs à tous les précédents avec une précision de 51 % et un rappel de 38 %.

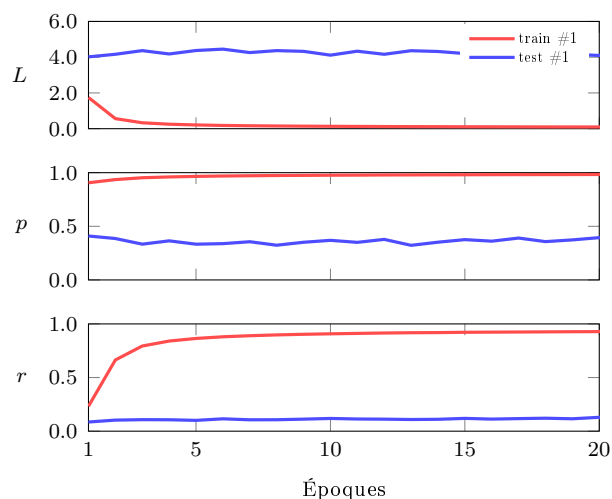


A.9 VGG 19 / YCB VIDEO / #1

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 21 bits
- 📁 Images :
 - 📄 Jeu de données : YCB VIDÉO
 - ↻ Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🍷 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 %
 - 📐 Recadrage centré sur l'objet à 384×384 pixels
 - 📏 Mise à l'échelle à 224×224
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 20

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0933	4.2006
précision	.75	0.9836	0.3817
rappel	.75	0.9294	0.1221
vrais négatifs	.25	0.9966	0.9545
vrais positifs	.75	0.9294	0.1221
vrais	.75	0.9959	0.9487
faux négatifs	.25	0.9793	0.2076
faux positifs	.75	0.9992	0.9901

Interprétation

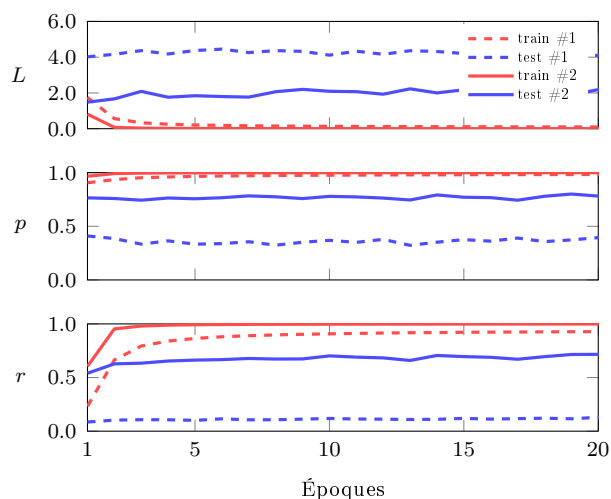
Les images recadrées à 384×384 incluent beaucoup d'autres objets en plus de celui annoté. Pendant l'entraînement plusieurs objets sont donc associés à la même étiquette ce qui est néfaste pour les performances et produit des faux positifs et des faux négatifs.

A.10 VGG 19 / YCB VIDEO / #2

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 21 bits
- 📁 Images :
 - 📂 Jeu de données : YCB VIDÉO
 - 🔄 Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🍷 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 %
 - 📐 Recadrage aux dimensions du rectangle encadrant plus une marge de 16 *px*
 - 🔍 Mise à l'échelle à 224×224
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 20

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0017	2.0579
précision	.75	0.9997	0.7738
rappel	.75	0.9990	0.6949
vrais négatifs	.25	0.9999	0.9837
vrais positifs	.75	0.9990	0.6949
vrais	.75	0.9999	0.9758
faux négatifs	.25	0.9996	0.7547
faux positifs	.75	0.9999	0.9898

Interprétation

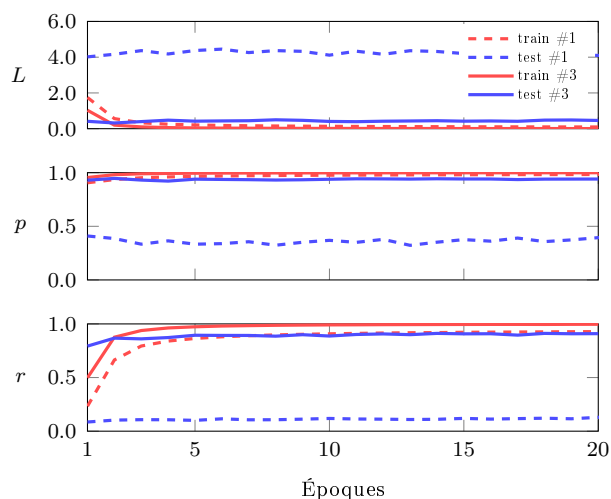
Recadrer les images aux dimensions des rectangles encadrants permet de limiter la présence des autres objets à l'image. De cette façon, les résultats obtenus sont bien meilleurs. Cela confirme que ce sont bien les objets distracteurs qui ont posés problème lors de l'entraînement précédent. Mais il y a encore beaucoup de faux négatifs.

A.11 VGG 19 / YCB VIDEO / #3

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 21 bits
- 📁 Images :
 - 📂 Jeu de données : YCB VIDÉO
 - 🔄 Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🍷 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 %
 - 📐 Recadrage aux dimensions du rectangle encadrant plus une marge de 16 *px*
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 20

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0055	0.4605
précision	.75	0.9992	0.9406
rappel	.75	0.9966	0.9092
vrais négatifs	.25	0.9998	0.9953
vrais positifs	.75	0.9966	0.9092
vrais	.75	0.9998	0.9929
faux négatifs	.25	0.9989	0.9324
faux positifs	.75	0.9999	0.9971

Interprétation

Effacer les arrière-plans permet d'effacer aussi les distracteurs. Cet entraînement donne de très bon résultats, même si on ne remplace pas les arrière-plans dans les images de tests. Cela confirme que ce sont bien les objets distracteurs qui ont posé problème lors des entraînements précédents.

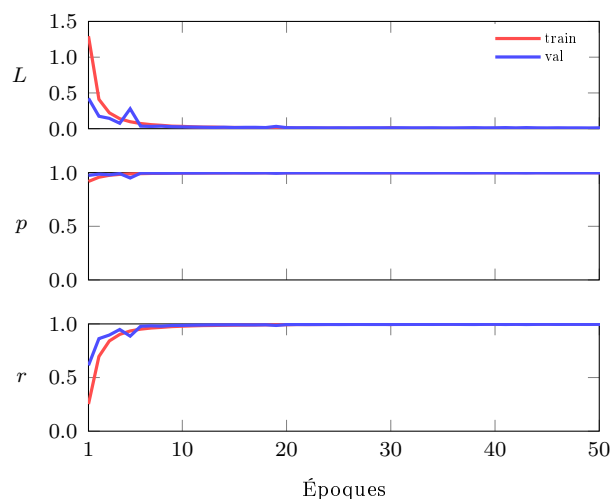


A.12 ALEXNET / LINEMOD

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : AlexNet
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : vecteur logique 15 bits
- 📁 Images :
 - 📄 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 📄 Préparation :
 - Recadrage à 227×227 px centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - Nombre d'époques : 1000

Entraînement











Scores

Mesure	τ	train	test
coût	-	$3.7e^{-5}$	0.01027
précision	.75	1	0.99855
rappel	.75	0.99995	0.99780
vrais négatifs	.25	0.99999	0.99977
vrais positifs	.75	0.99995	0.99780
vrais	.75	0.99999	0.99966
faux négatifs	.25	1	0.99845
faux positifs	.75	1	0.99985









Interprétation

Très bons résultats. Moins bons que VGG 19, mais ils sont obtenus avec un réseau plus simple, dont le temps d'entraînement est plus court.

Résultats

							
ape (100 %)	can (100 %)	hole (100 %)	iron (100 %)	glue (100 %)	phone (100 %)	vise (100 %)	duck (100 %)

Résultats corrects

							
phone/can (77 %)	cat/duck (94 %)	phone/hole (99 %)	cat/can (73 %)	can/lamp (100 %)	glue/lamp (62 %)	can/glue (85 %)	lamp/phone (83 %)

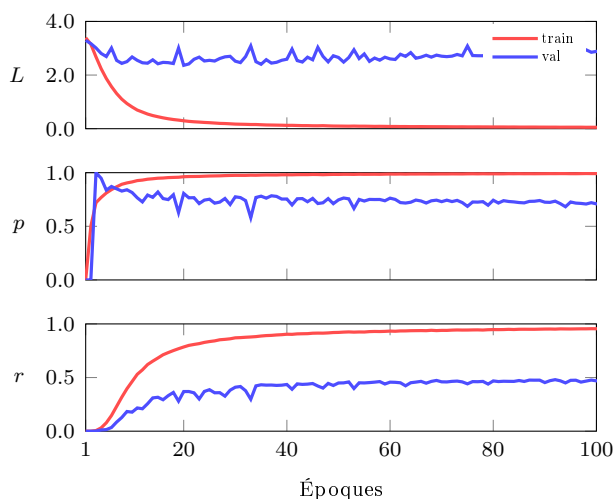
Résultats incorrects

A.13 ALEXNET / TLESS

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : VGG 19
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : vecteur logique 30 bits
- 📁 Images :
 - 📄 Jeu de données : TLESS
 - ↻ Répartition : 37584 entraînement et 69552 validation/tests
 - 🍹 Préparation :
 - 📄 Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - 📄 Recadrage aux dimensions du rectangle encadrant plus une marge de $16px$, fois un facteur aléatoire d'échelle compris entre 0.95 et 1.05
 - 📄 Mise à l'échelle à 227×227
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📄 Nombre d'époques : 100

Entraînement

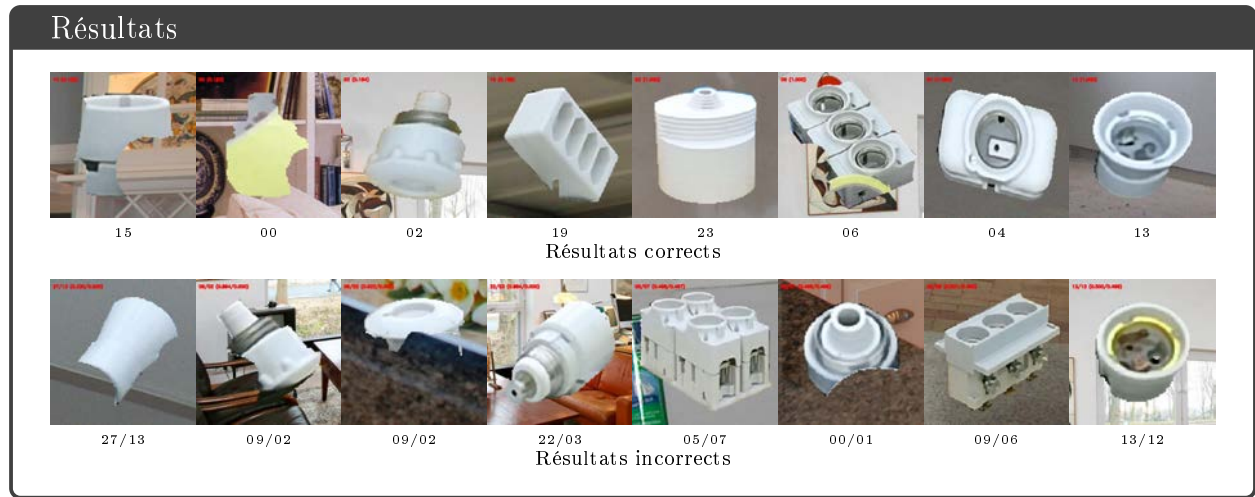


Scores

Mesure	τ	train	test
coût	-	0.0497	2.8307
précision	.75	0.9916	0.7163
rappel	.75	0.9574	0.4772
vrais négatifs	.25	0.9986	0.9802
vrais positifs	.75	0.9574	0.4772
vrais	.75	0.9983	0.9762
faux négatifs	.25	0.9909	0.5758
faux positifs	.75	0.9997	0.9934

Interprétation

Résultats meilleurs que VGG 19 à moindre coût.



A.14 ALEXNET / YCB VIDEO

Implémentation

Modèle :

- ⊗ Réseau : AlexNet
- ↔ Entrée : $227 \times 227 \times 3$
- ↳ Sortie : vecteur logique 21 bits

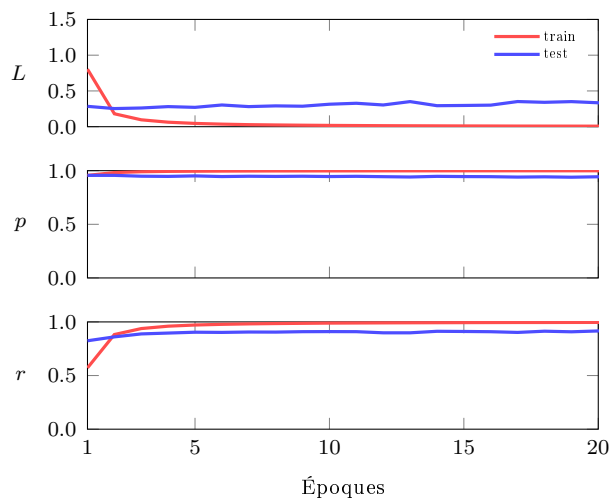
Images :

- 📁 Jeu de données : YCB VIDÉO
- ↻ Répartition : 515 350 entraînement et 97 242 validation/tests
- 🍷 Préparation :
 - ∇ Filtration des objets occultés à plus de 70 %
 - 📐 Recadrage aux dimensions du rectangle encadrant plus une marge de 16 *px*
 - 🔍 Mise à l'échelle à 227×227
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement

Descente de gradient :

- ⊗ Nombre d'époques : 20

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.0064	0.3550
précision	.75	0.9992	0.9409
rappel	.75	0.9959	0.9094
vrais négatifs	.25	0.9998	0.9952
vrais positifs	.75	0.9959	0.9094
vrais	.75	0.9997	0.9929
faux négatifs	.25	0.9988	0.9339
faux positifs	.75	0.9999	0.9971

Interprétation

Résultats légèrement meilleurs que VGG 19.

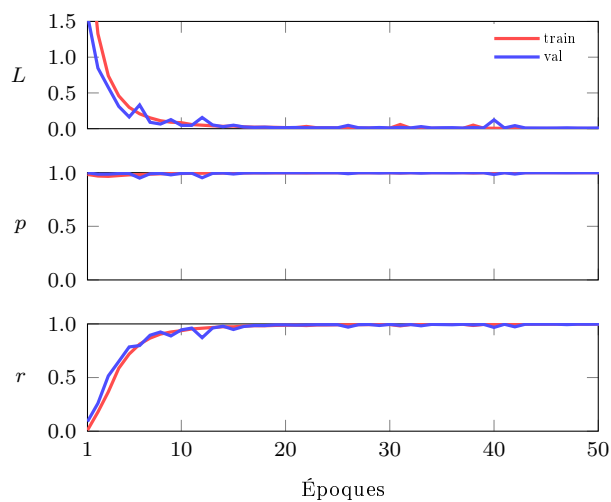


A.15 GOOGLNET / LINEMOD

Implémentation

- 🏠 Modèle :
 - 🔗 Réseau : AlexNet
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 15 bits
- 📁 Images :
 - 📂 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 🍷 Préparation :
 - Recadrage à 224×224 px centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - Nombre d'époques : 1000

Entraînement











Scores

Mesure	τ	train	val
coût	-	$2.38e^{-5}$	0.0075
précision	0.75	1.0	0.9995
rappel	0.75	1.0	0.9991
vrais négatifs	0.25	1.0	0.9999
vrais positifs	0.75	1.0	0.9991
vrais	0.75	1.0	0.9999
faux négatifs	0.25	1.0	0.9994
faux positifs	0.75	1.0	0.9999









Interprétation

Résultats meilleurs que ceux de VGG et AlexNet sur le même jeu de données, préparé de la même manière.

Résultats

							
ape (100 %)	cat (100 %)	vise (100 %)	driller (100 %)	lamp (100 %)	glue (100 %)	can (100 %)	phone (100 %)

Résultats corrects

							
glue/lamp (100 %)	lamp/glue (97 %)	lamp/glue (99 %)	lamp/glue (99 %)	phone/glue (42 %)	lamp/glue (98 %)	phone/glue (75 %)	cat/phone (76 %)

Résultats incorrects

A.16 GOOGLNET / TLESS

Implémentation

Modèle :

- ∞ Réseau : GoogLeNet
- ↔ Entrée : $224 \times 224 \times 3$
- ↳ Sortie : vecteur logique 30 bits

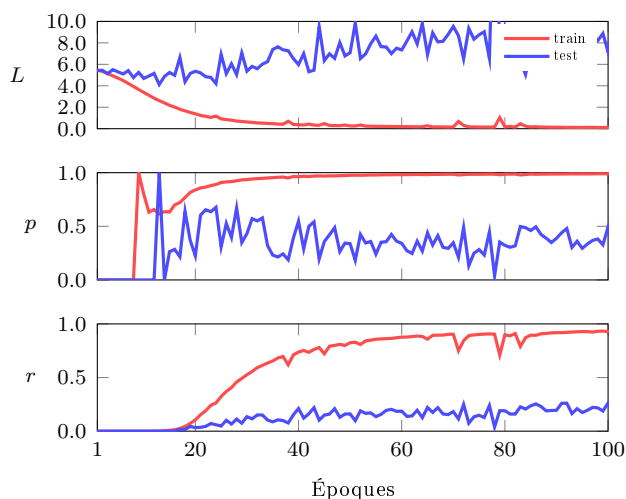
Images :

- 📄 Jeu de données : TLESS
- ↻ Répartition : 37584 entraînement et 69552 validation/tests
- 🍹 Préparation :
 - ∇ Filtration des objets occultés à plus de 70 % ce qui produit 62681 images de tests
 - 📐 Recadrage aux dimensions du rectangle encadrant plus une marge de $16px$, fois un facteur aléatoire d'échelle compris entre 0.95 et 1.05
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement

Descente de gradient :

- ☒ Nombre d'époques : 100

Entraînement



Scores

Mesure	τ	train	test
coût	-	0.1093	9.5652
précision	.75	0.9902	0.2836
rappel	.75	0.9336	0.1517
vrais négatifs	.25	0.9978	0.9667
vrais positifs	.75	0.9336	0.1517
vrais	.75	0.9974	0.9589
faux négatifs	.25	0.9887	0.2565
faux positifs	.75	0.9996	0.9867

Interprétation

Résultats inférieurs à ceux de VGG 19 et AlexNet sur le même jeu de données, préparé de la même manière.

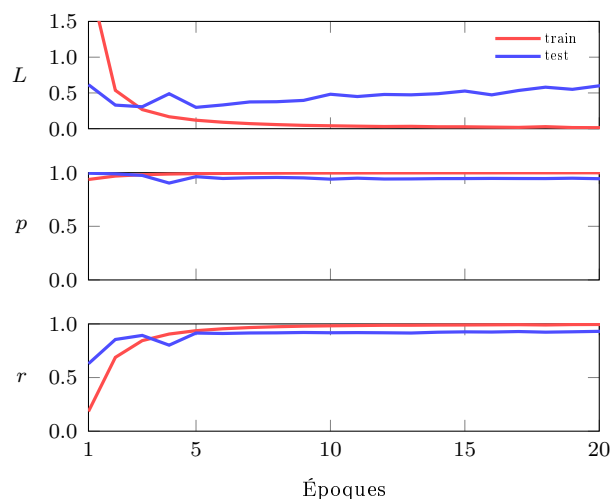


A.17 GOOGLNET / YCB VIDEO

Implémentation

- 🏠 Modèle :
 - 🔗 Réseau : GoogLeNet
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : vecteur logique 21 bits
- 📁 Images :
 - 📂 Jeu de données : YCB VIDÉO
 - 🔄 Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🍷 Préparation :
 - 📉 Filtration des objets occultés à plus de 70 %
 - 📐 Recadrage aux dimensions du rectangle encadrant plus une marge de $16px$
 - 🔍 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📦 Nombre d'époques : 20

Entraînement

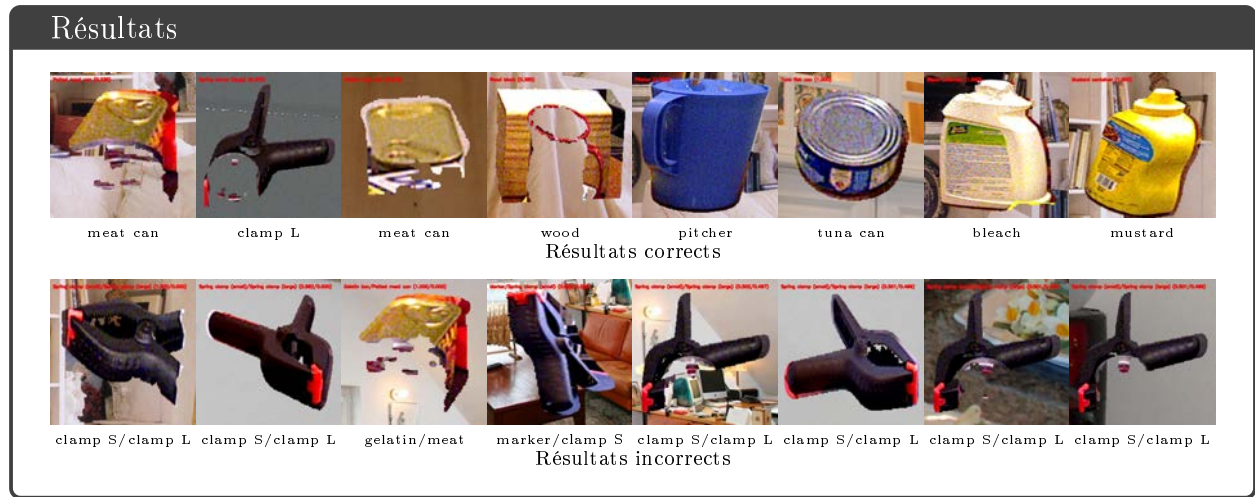


Scores

Mesure	τ	train	test
coût	-	0.0136	0.5920
précision	.75	0.9989	0.9445
rappel	.75	0.9945	0.9317
vrais négatifs	.25	0.9997	0.9965
vrais positifs	.75	0.9945	0.9317
vrais	.75	0.9996	0.9941
faux négatifs	.25	0.9984	0.9426
faux positifs	.75	0.9999	0.9972

Interprétation

Résultats légèrement meilleurs que ceux de VGG 19 sur le même jeu de données, préparé de la même manière.

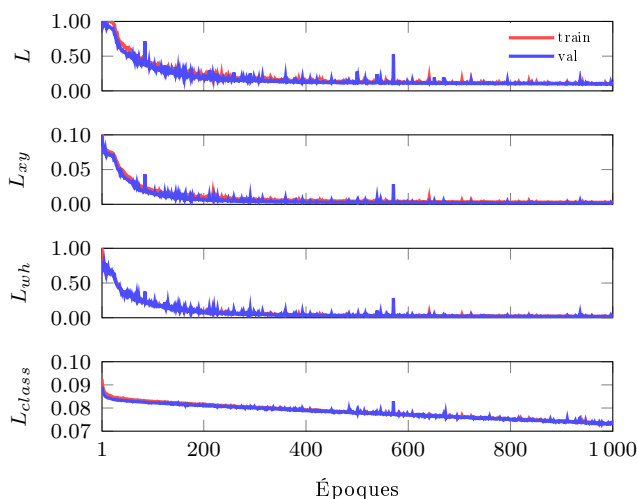


A.18 YOLO / LINEMOD

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : YOLO
 - ⚙️ Paramètres : $S = 1, C = 15, B = 2$
 - 📏 Entrée : $640 \times 480 \times 3$
 - 📏 Sortie : $1 \times 1 \times (15 + 2 \times 5)$
 - 📈 Activation : Linéaire
- 📁 Images :
 - 📦 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 🔧 Préparation :
 - 📄 Remplacement de l'arrière-plan avec une image de SUN
- 📉 Descente de gradient :
 - 🕒 Fonction de coût : YOLO (somme des carrées des différences)
 - 📅 Nombre d'époques : 1000

Entraînement



Scores

Mesure	train	test
coût	0.1059	0.0988
coût no obj	0.0031	0.0015
coût obj	0.0057	0.0032
coût class	0.0735	0.0734
coût wh	0.0206	0.0189
coût xy	0.0030	0.0018

Interprétation

Les coûts L_{xy} ; L_{obj} et L_{noobj}) ont bien diminué. YOLO apprend donc bien à localiser les objets. Le coût L_{wh} est moins bon, de même pour L_{class} . L'imprécision de dimensionnement est due à la mauvaise spécialisation des différents détecteurs de la cellules ($B = 2$). Nous avons remarqué visuellement qu'ils ont tous les deux appris à prédire les mêmes ratios d'aspect. Les mauvais résultats de classification sont dus à l'utilisation du coût L_2 . Ces problèmes sont résolus dans YOLO V2 grâce à la cross-entropie catégorique et aux ancres.

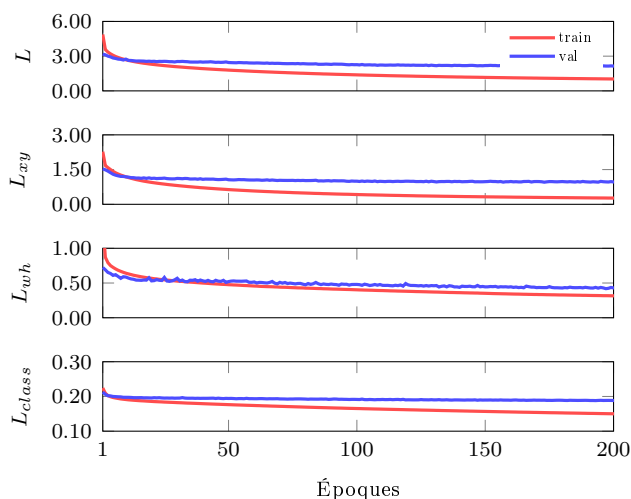


A.19 YOLO / YCB VIDEO

Implémentation

- 📦 Modèle :
 - 🔗 Réseau : YOLO
 - ⚙️ Paramètres : $S = 5$, $C = 21$, $B = 2$
 - 📏 Entrée : $640 \times 480 \times 3$
 - 📏 Sortie : $5 \times 5 \times (21 + 2 \times 5)$
 - 📡 Activation : Linéaire
- 🖼️ Images :
 - 📦 Jeu de données : YCB VIDÉO
 - 🔄 Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🔧 Préparation :
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 🕒 Fonction de coût : YOLO (somme des carrées des différences)
 - 📦 Nombre d'époques : 200

Entraînement



Scores

Mesure	train	test
coût	1.0257	2.1332
coût no obj	0.2012	0.3782
coût obj	0.0913	0.1774
coût class	0.1499	0.1885
coût wh	0.3151	0.4299
coût xy	0.2679	0.9584

Interprétation

Les mêmes objets sont visibles dans une séquence vidéo ce qui donne des répétitions de dispositions d'objets. Ces répétitions ne permettent pas à YOLO d'apprendre correctement à prédire les rectangles encadrants. Les dispositions d'objets étant différentes dans les images de test, YOLO ne généralise pas bien sur ces images.

Résultats



Résultats corrects



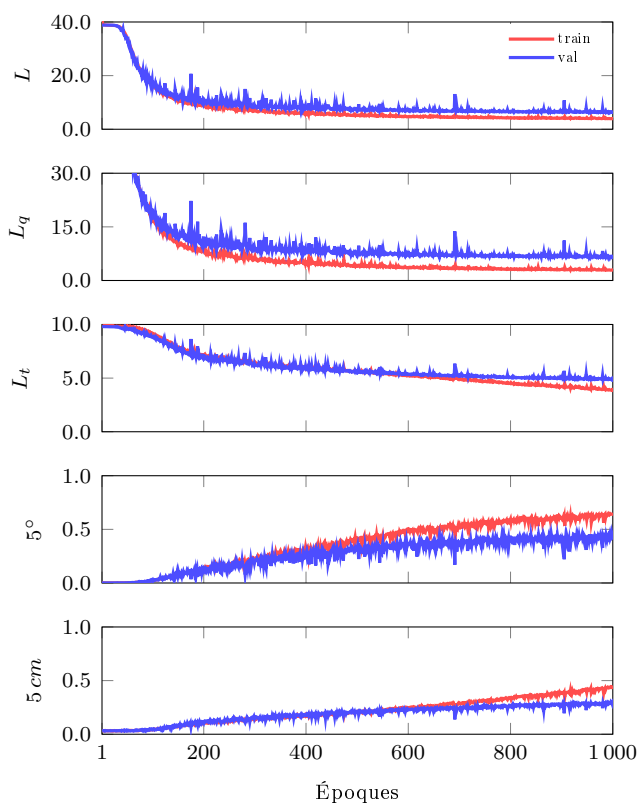
Résultats incorrects

A.20 POSENET / LINEMOD / #1

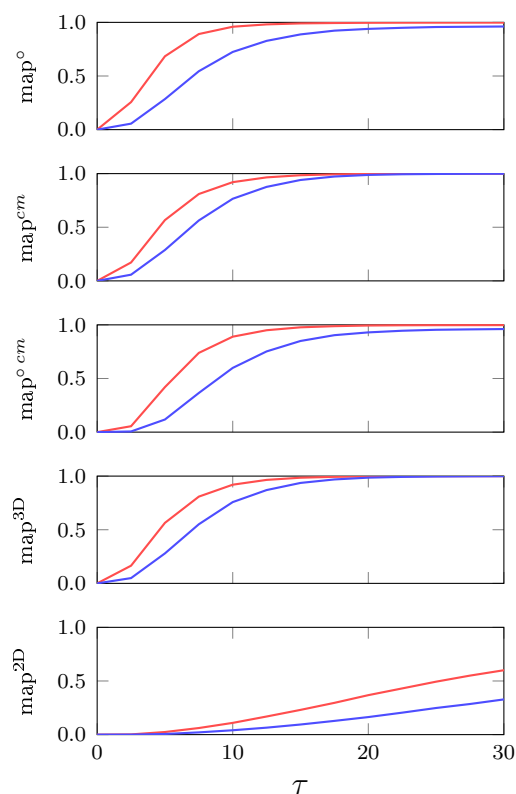
Implémentation

- 📦 Modèle :
 - 🔗 Réseau : PoseNet (GoogLeNet)
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : quaternion 4 et vecteur de translation 3
 - 📈 Activation : Linéaire
- 📁 Images :
 - 📦 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation (20034)
 - 🔧 Préparation :
 - Recadrage à 227×227 px centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 🕒 Fonction de coût : distance euclidienne
 - 📦 Nombre d'époques : 1000

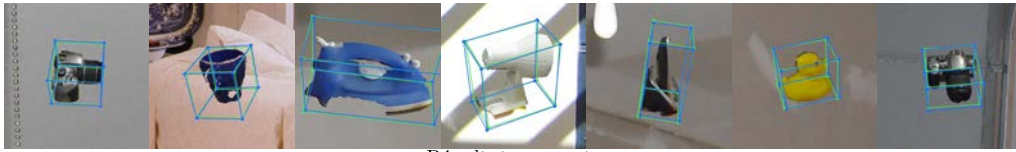
Entraînement



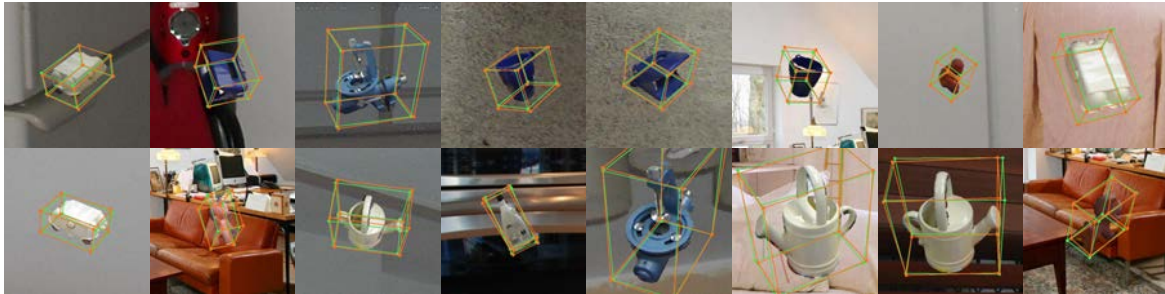
Scores



Résultats



Résultats corrects



Résultats incorrects

Interprétation

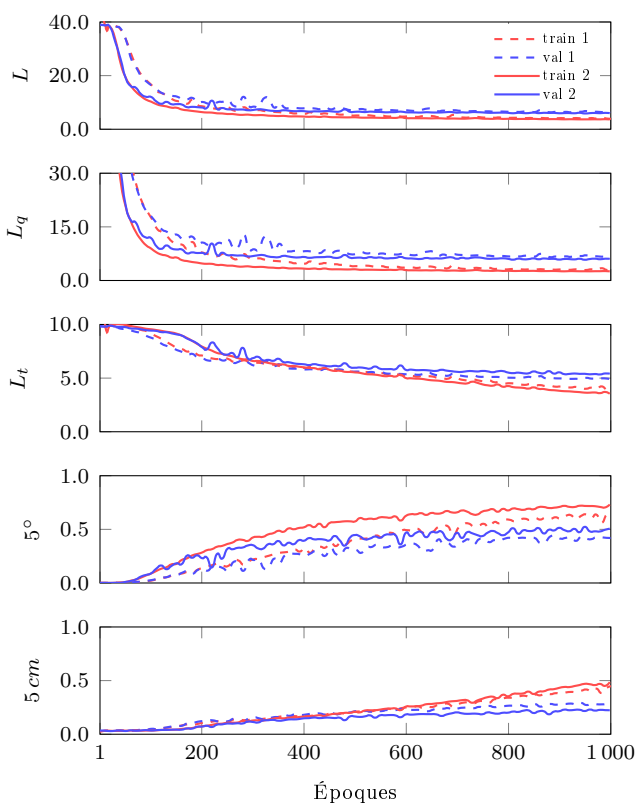
Fonction de coût bien minimisée mais résultats limités. On remarque que malgré les bons résultats de précisions à 5° , 5 cm et $5^\circ 5\text{ cm}$, la précision en 2D sur l'image est peu satisfaisante.

A.21 POSENET / LINEMOD / #2

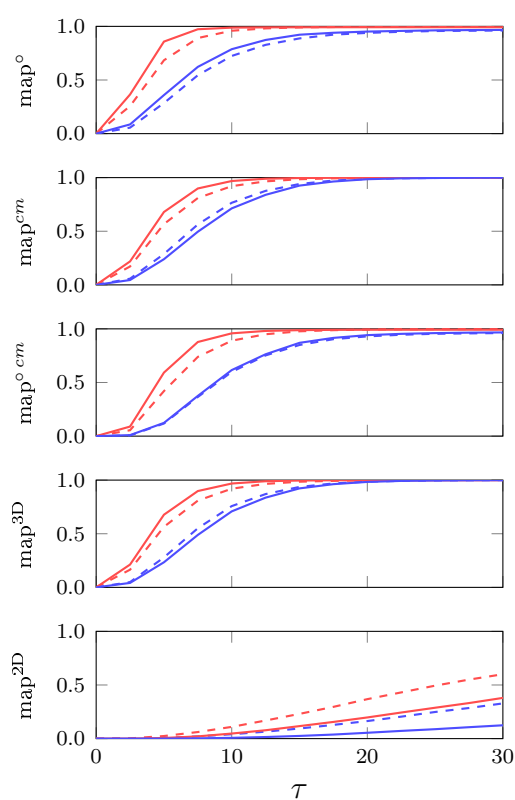
Implémentation

- 📦 Modèle :
 - 🔗 Réseau : PoseNet (GoogLeNet)
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : quaternion 4 et vecteur de translation 3
 - 📄 Activation : Linéaire
- 📁 Images :
 - 📄 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 📄 Préparation :
 - 📄 Recadrage à la taille du rectangle encadrant plus une marge de 16 pixels
 - 📄 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📄 Fonction de coût : distance euclidienne
 - 📄 Nombre d'époques : 1000

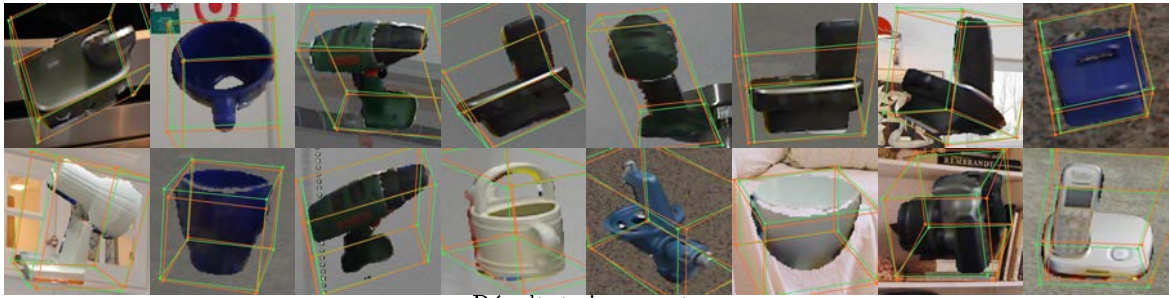
Entraînement



Scores



Résultats



Résultats incorrects

Interprétation

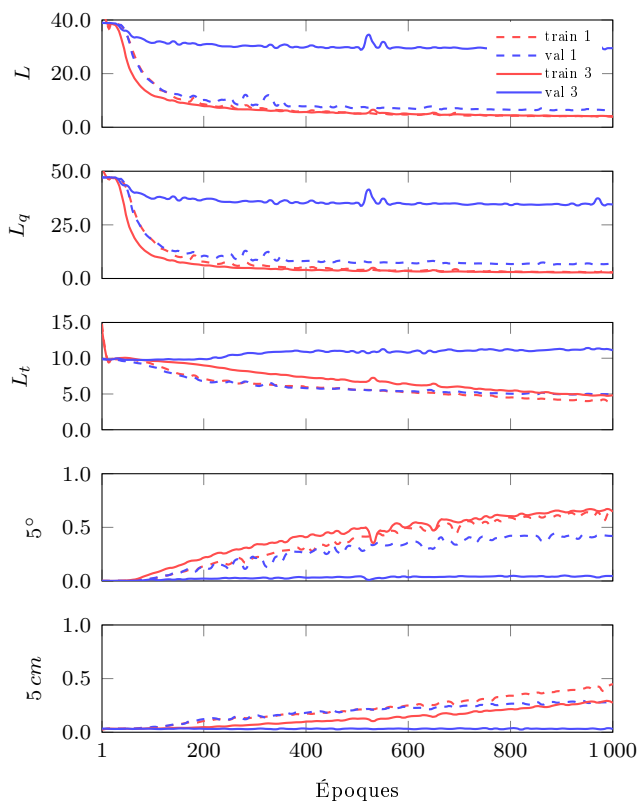
La fonction de coût a une valeur très proche de celle du précédent entraînement. Mais on constate un écart plus important entre entraînement et validation, donc une moins bonne généralisation. Les résultats sont bien moins bons. Ce qui est logique car redimensionner aux dimensions des rectangles encadrant signifie que l'on redimensionne à des tailles différentes les même objets. Or ceci est embigüe pour PoseNet car il pourrait aussi sagir d'un changement de distance à l'objet. On voit ici le principale problème de PoseNet : on peut utiliser très peu d'augmentations d'images.

A.22 POSENET / LINEMOD / #3

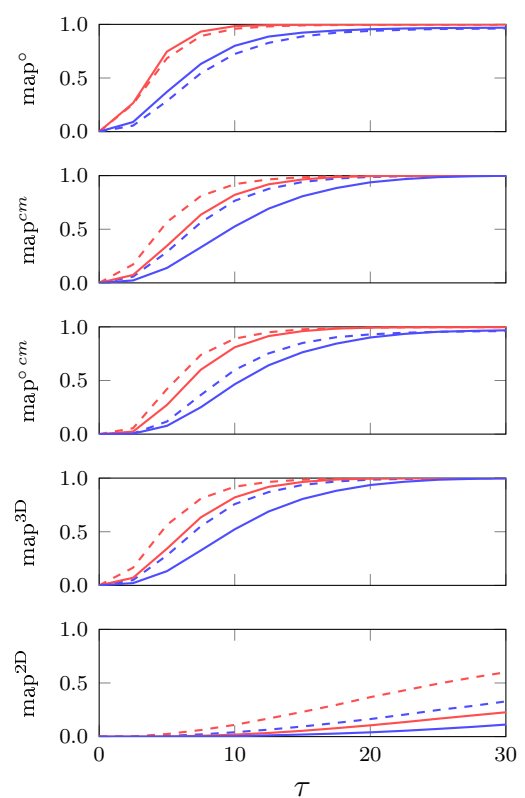
Implémentation

- 📦 Modèle :
 - 🔗 Réseau : PoseNet (GoogLeNet)
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : quaternion 4 et vecteur de translation 3
 - 📈 Activation : Linéaire
- 📁 Images :
 - 📦 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation (20034)
 - 📄 Préparation :
 - 📏 Recadrage aux dimensions du rectangle encadrant plus une marge de $16px$, fois un facteur aléatoire d'échelle compris entre 0.95 et 1.05
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation
- 📉 Descente de gradient :
 - 💰 Fonction de coût : distance euclidienne
 - 📦 Nombre d'époques : 1000

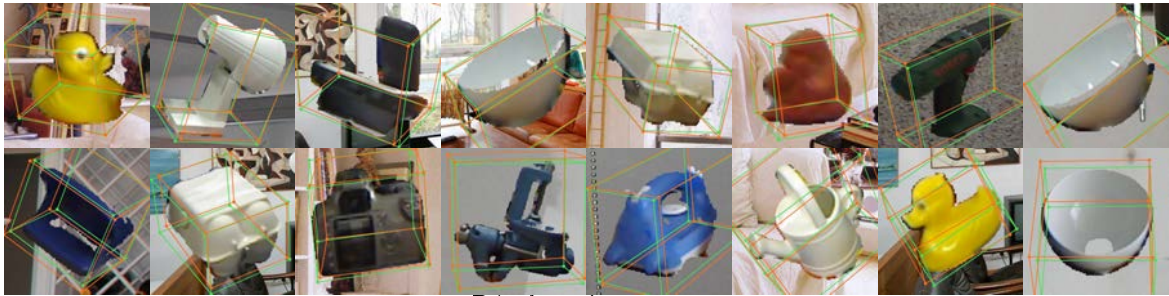
Entraînement



Scores



Résultats



Résultats incorrects

Interprétation

Encore moins bon car l'augmentation d'image appliqué est incompatible avec les étiquettes de PoseNet. En particulier la variation aléatoire de taille est ambiguë avec les étiquettes de translations. Ce qui est confirmé par la valeur du terme de la fonction de coût consacré à la translation.

A.23 POSENET / T-LESS

Implémentation

📦 Modèle :

🔗 Réseau : PoseNet (GoogLeNet)

📄 Entrée : $227 \times 227 \times 3$

📄 Sortie : quaternion 4 et vecteur de translation 3

📄 Activation : Linéaire

📁 Images :

📄 Jeu de données : TLESS

🔄 Répartition : 37584 entraînement et 69552 validation/tests

🔧 Préparation :

📄 Filtration des objets occultés à plus de 70 %

📄 Recadrage à 227×227 pixels centré sur l'objet

😊 Remplacement de l'arrière-plan avec une image de SUN

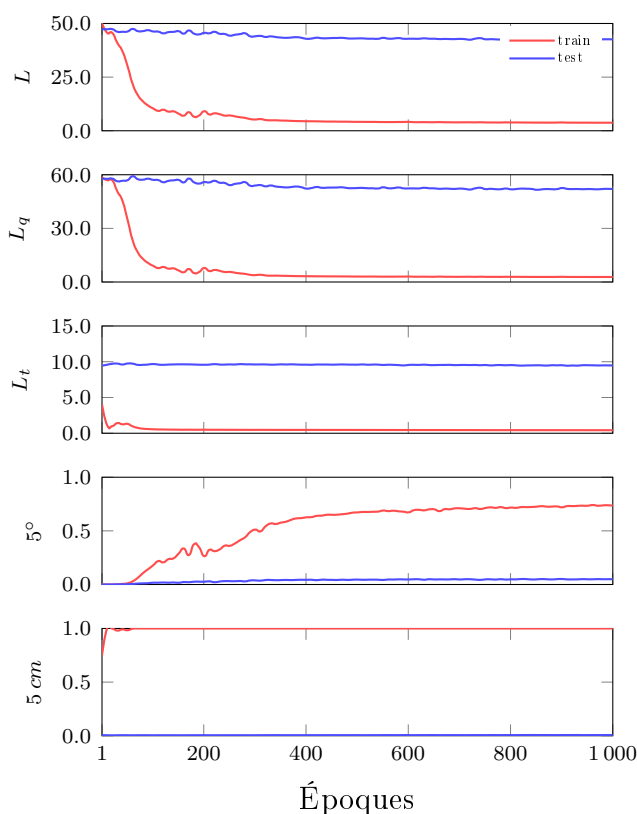
💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement

📉 Descente de gradient :

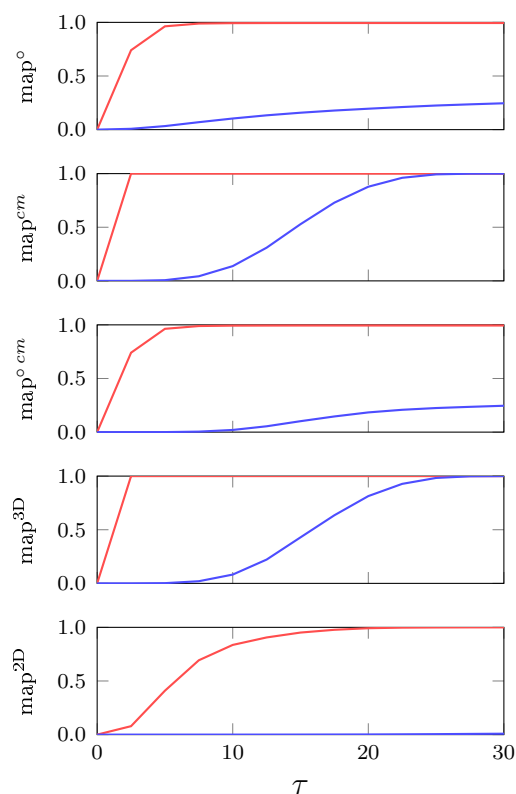
📄 Fonction de coût : distance euclidienne

📄 Nombre d'époques : 1000

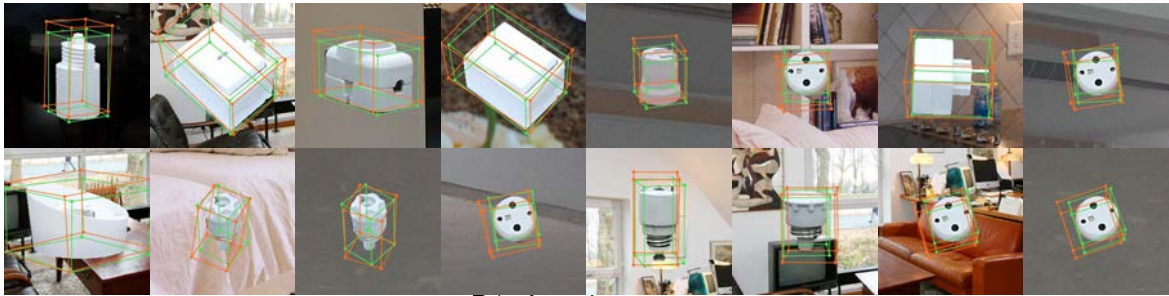
Entraînement



Scores



Résultats



Résultats incorrects

Interprétation

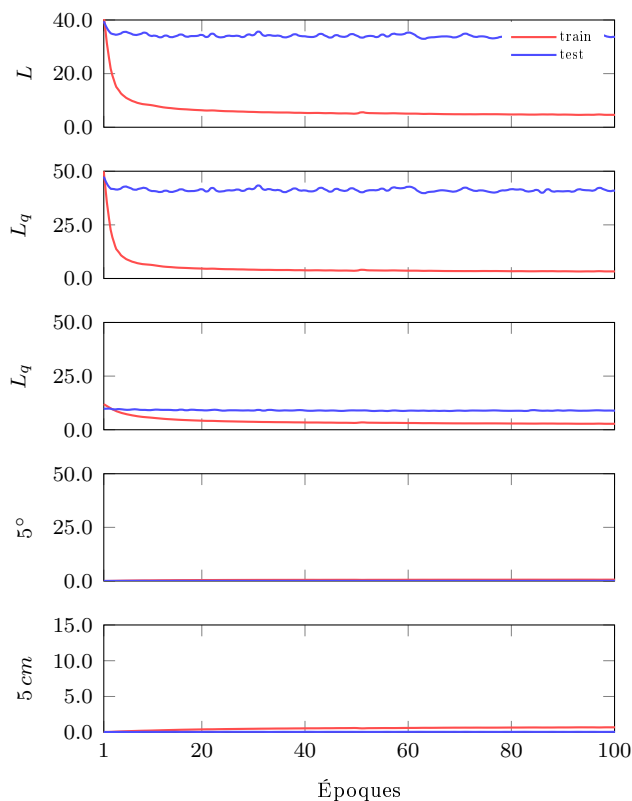
Résultat insatisfaisant. La rotation progresse difficilement pendant l'entraînement et à la fin les résultats ne sont pas bons à cause des ambiguïtés de pose des objets (symétrie, géométrie répétitive). Résultats inutilisable sur les images de tests.

A.24 POSENET / YCB-VIDEO

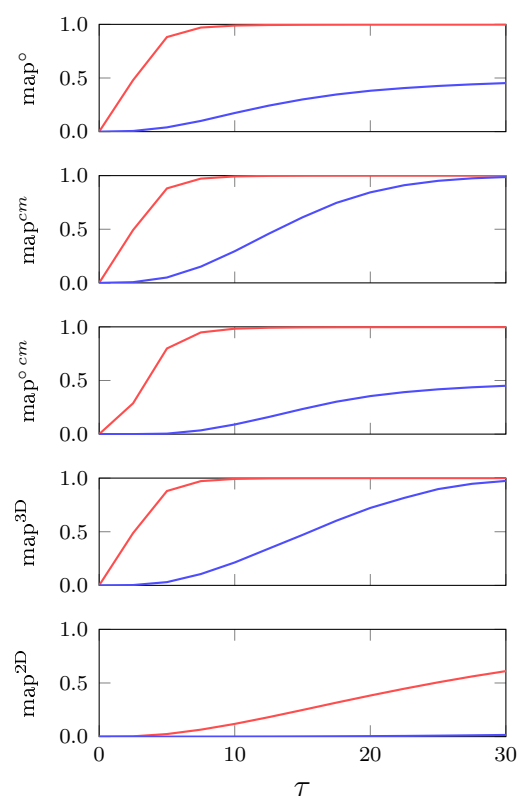
Implémentation

- 🏠 Modèle :
 - 🔗 Réseau : PoseNet (GoogLeNet)
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : quaternion 4 et vecteur de translation 3
 - 📄 Activation : Linéaire
- 🖼 Images :
 - 📄 Jeu de données : YCB VIDÉO
 - 🔄 Répartition : 515 350 entraînement et 97 242 validation/tests
 - 🔧 Préparation :
 - 📄 Filtration des objets occultés à plus de 70 %
 - 📄 Recadrage à 227×227 pixels centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 Descente de gradient :
 - 📄 Fonction de coût : distance euclidienne
 - 📄 Nombre d'époques : 100

Entraînement



Scores



Résultats



Résultats incorrects

Interprétation

Dans cette expérience nous présentons chaque instance d'objet individuellement à PoseNet. PoseNet apprend relativement bien à prédire la pose des objets comme le montre les mesures de recalage 3D sur les images d'entraînement. Les résultats de recalage 2D sur les mêmes images ne sont pas très bons. PoseNet ne généralise aussi pas bien sur les images de test car les images de test présentent les objets dans des poses différentes de celles des images d'entraînement.

A.25 BB8 / LINEMOD / FAIL

Implémentation

📦 Modèle :

🔗 Réseau : BB8 (VGG-19)

📄 Entrée : $224 \times 224 \times 3$

📄 Sortie : $15 \times 8 \times 2$

📈 Activation : Linéaire

📁 Images :

📄 Jeu de données : LINEMOD

🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)

🔧 Préparation :

📐 Recadrage à $227 \times 227 px$ centré sur l'objet

📏 Redimensionnage à $224 \times 224 px$

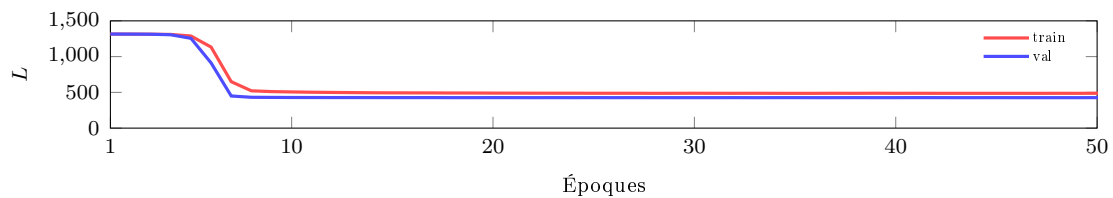
😊 Remplacement de l'arrière-plan avec une image de SUN

📉 Descente de gradient :

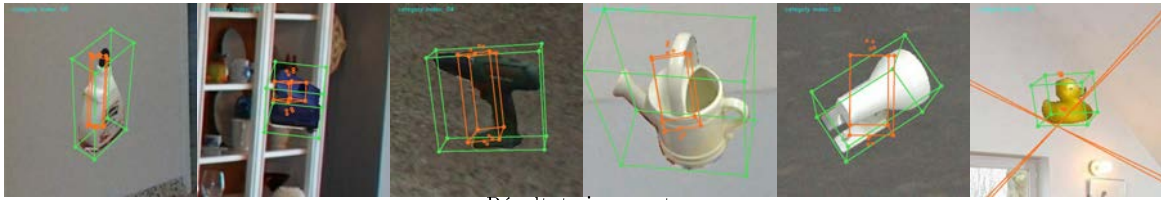
🕒 Fonction de coût : distance euclidienne

📦 Nombre d'époques : 1000

Entraînement



Résultats



Résultats incorrects

Interprétation

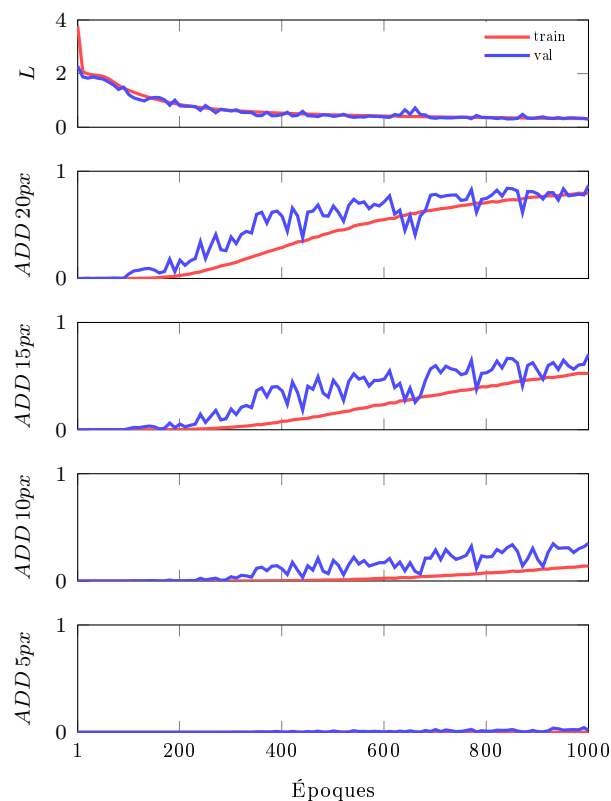
BB8 n'est pas capable d'apprendre à prédire la pose des objets de LINEMOD avec des points dont les coordonnées sont exprimés en pixels.

A.26 BB8 / LINEMOD

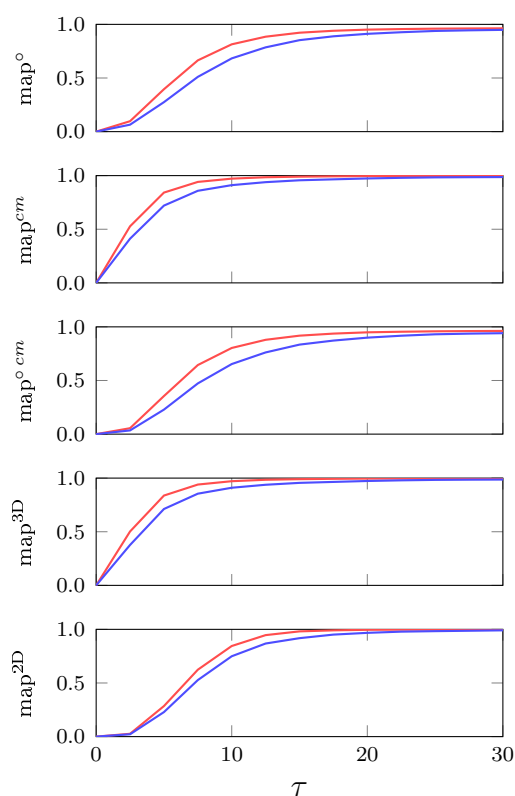
Implémentation

- 📦 Modèle :
 - 🔗 Réseau : BB8 (VGG-19)
 - 📄 Entrée : $224 \times 224 \times 3$
 - 📄 Sortie : $15 \times 8 \times 2$
 - 📈 Activation : Linéaire
- 📁 Images :
 - 📦 Jeu de données : LINEMOD
 - 🔄 Répartition : 75 % entraînement (60048) et 25 % validation/tests (20034)
 - 🔧 Préparation :
 - 📐 Recadrage à 227×227 px centré sur l'objet
 - 🔄 Redimensionnage à 224×224 px
 - 📏 Normalisation des coordonnées 0 et 1
 - 😊 Remplacement de l'arrière-plan avec une image de SUN
 - 🌈 Variations aléatoires de luminosité, contraste, teinte et saturation
- 📉 Descente de gradient :
 - 📌 Fonction de coût : distance euclidienne
 - 📅 Nombre d'époques : 1000

Entraînement



Scores



Résultats



Résultats corrects



Résultats incorrects

Interprétation

BB8 apprend correctement à prédire la pose des objets de LINEMOD avec des points dont les coordonnées sont normalisées entre 0 et 1.

A.27 BB8 / T-LESS

Implémentation

📦 Modèle :

🔗 Réseau : BB8 (VGG-19)

📏 Sortie : $15 \times 8 \times 2$

📐 Entrée : $224 \times 224 \times 3$

🔌 Activation : Linéaire

🖼️ Images :

📁 Jeu de données : TLESS

🔄 Répartition : 37584 entraînement et 69552 validation/tests

🍹 Préparation :

📉 Filtration des objets occultés à plus de 70%

📐 Recadrage à 227×227 pixels centré sur l'objet

🔄 Redimensionnage à 224×224 px

📏 Normalisation des coordonnées 0 et 1

😊 Remplacement de l'arrière-plan avec une image de SUN

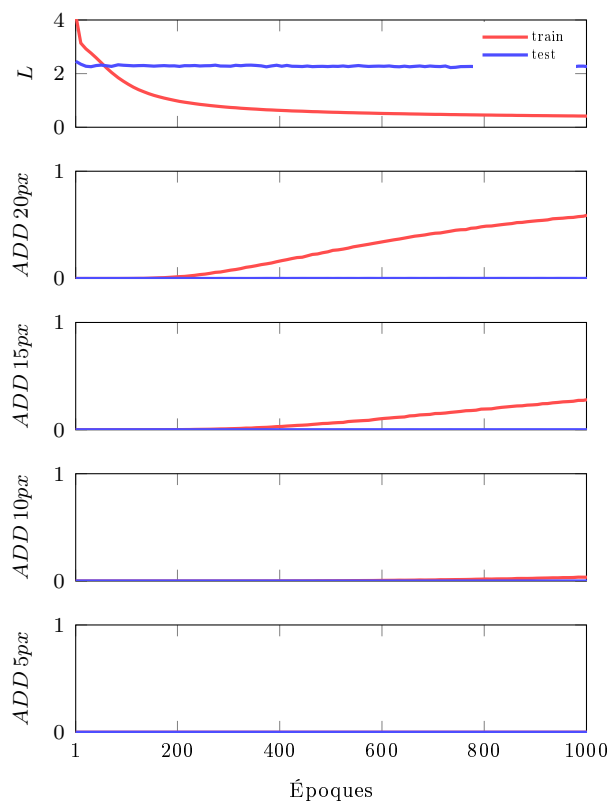
💡 Variations aléatoires de luminosité, contraste, teinte et saturation

📉 Descente de gradient :

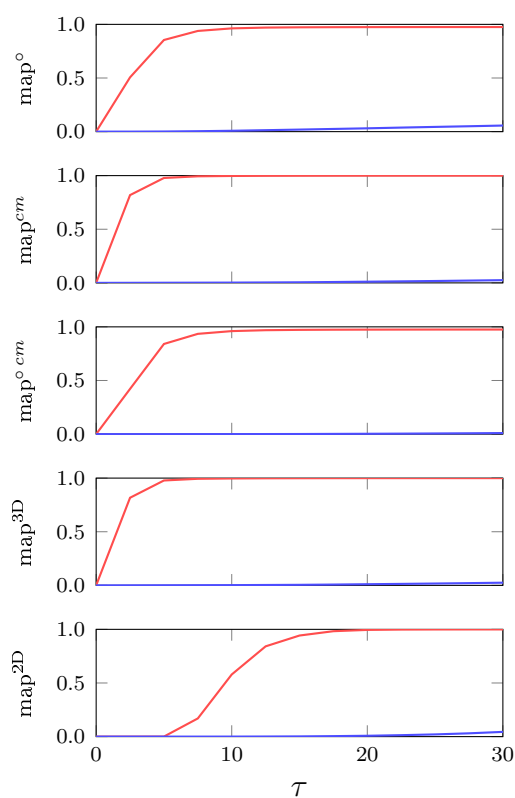
📌 Fonction de coût : distance euclidienne

📦 Nombre d'époques : 1000

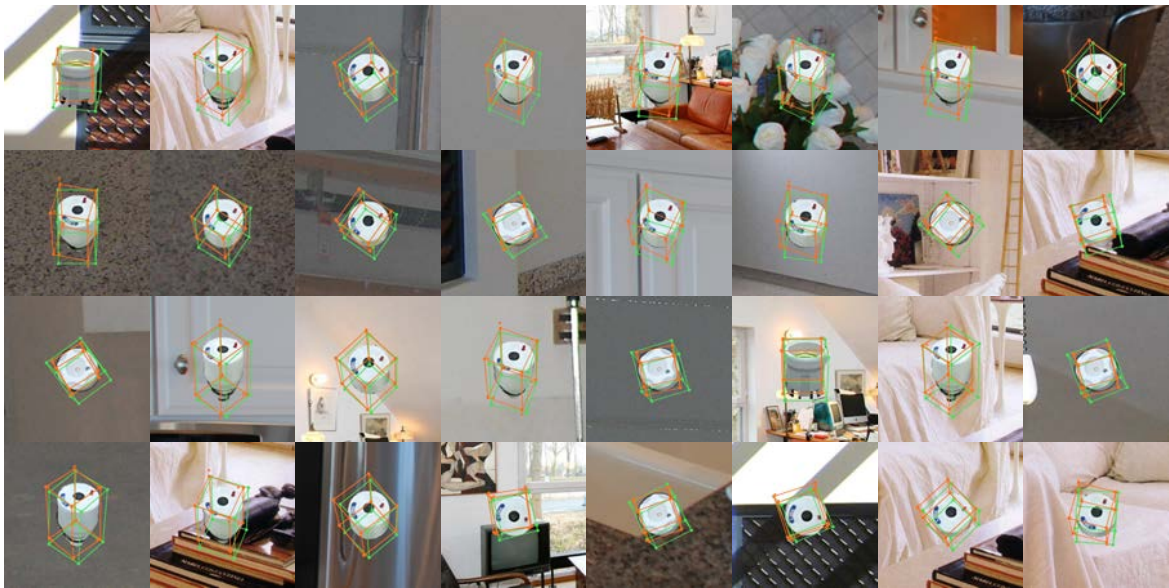
Entraînement



Scores



Résultats



Résultats incorrects

Interprétation

BB8 apprend correctement à prédire la pose dans les images d'entraînement mais ne généralise pas sur les image de test qui sont beaucoup plus complexes. Et ce malgré le fait que nous avons éliminé les occlusions et les distracteurs en effaçant les arrières-plans des images de tests.

A.28 BB8 / YCB-VIDEO

Implémentation

📦 Modèle :

🔗 Réseau : BB8 (VGG-19)

📄 Sortie : $21 \times 8 \times 2$

📷 Entrée : $224 \times 224 \times 3$

🔌 Activation : Linéaire

📁 Images :

📀 Jeu de données : YCB VIDÉO

🔄 Répartition : 515 350 entraînement et 97 242 validation

🍷 Préparation :

📉 Filtration des objets occultés à plus de 70%

📐 Recadrage à 227×227 pixels centré sur l'objet

🔄 Redimensionnage à 224×224 px

📏 Normalisation des coordonnées 0 et 1

😊 Remplacement de l'arrière-plan avec une image de SUN

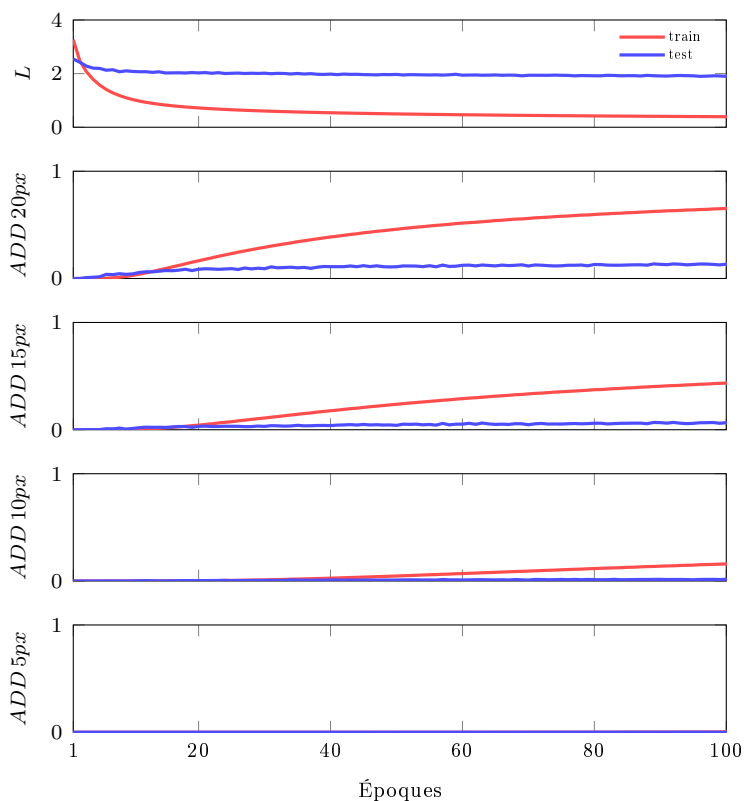
🌟 Luminosité, contraste, teinte et saturation aléatoires (training)

📉 Descente de gradient :

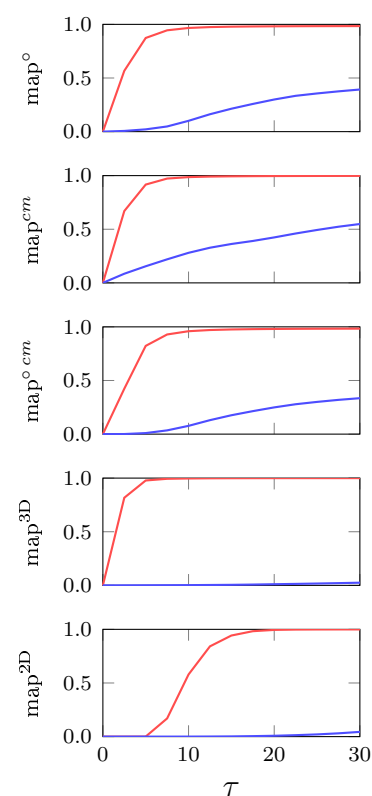
📈 Fonction de coût : distance euclidienne

📦 Nombre d'époques : 100

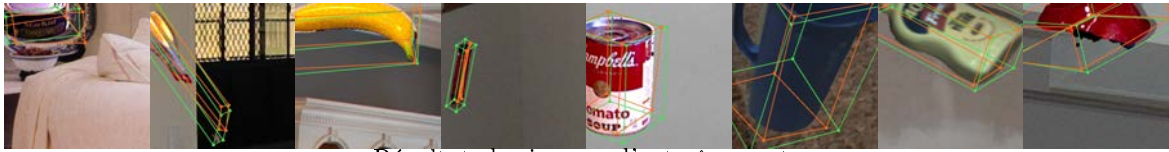
Entraînement



Scores



Résultats



Résultats les images d'entraînements

Interprétation

Les résultats sont encourageants sur les images d'entraînement mais très mauvais sur les images de test. Avec les images d'entraînement à notre disposition, BB8 ne généralise pas sur des images avec des poses nouvelles et des assortiments d'objets inédits.

A.29 POSENET / NEMA / #1 (arrière-plans SUN)

Implémentation

🏠 Modèle :

🔗 Réseau : PoseNet (GoogLeNet)

📄 Sortie : $4 + 3 (q + t)$

📄 Entrée : $227 \times 227 \times 3$

🔗 Activation : Linéaire

📁 Images :

📁 Jeu de données : NEMA

🔄 Répartition : 132 480 entraînement et 40 320 validation/tests

🍷 Préparation :

📐 Recadrage à 227×227 pixels centré sur l'objet

😊 Remplacement de l'arrière-plan avec une image de SUN

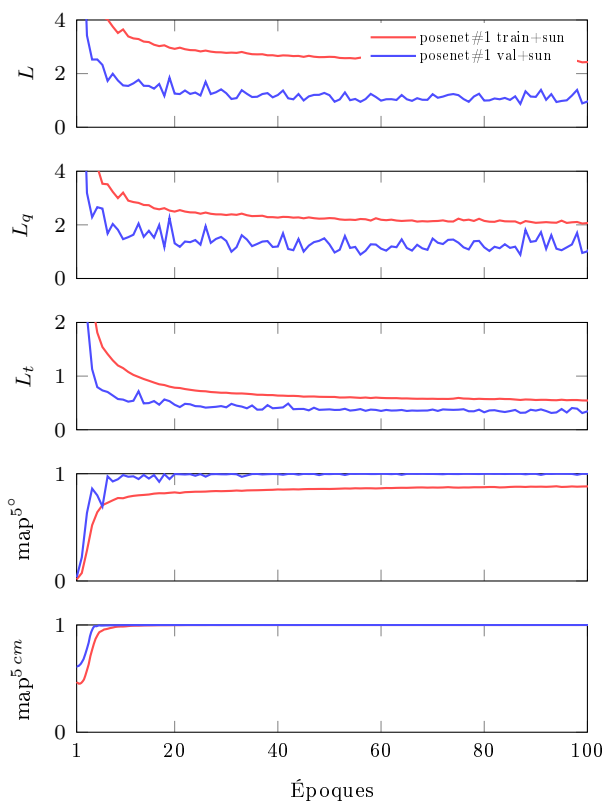
💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement

📉 Descente de gradient :

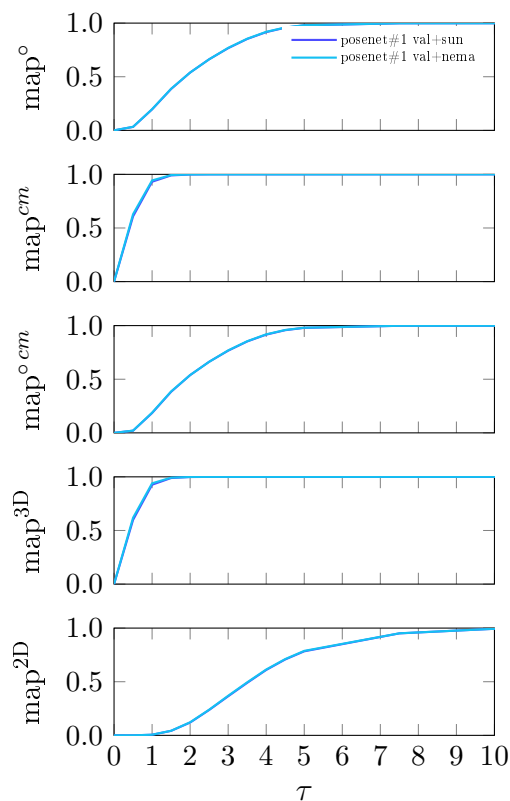
🕒 Fonction de coût : distance euclidienne

📅 Nombre d'époques : 100

Entraînement



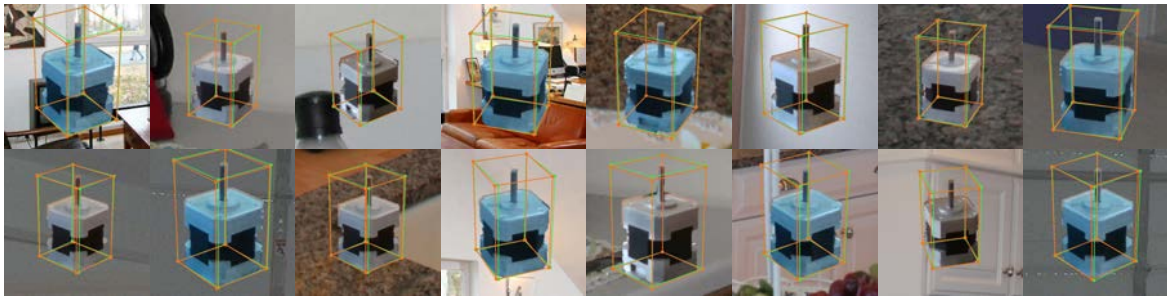
Scores



Résultats



Résultats corrects



Résultats incorrects

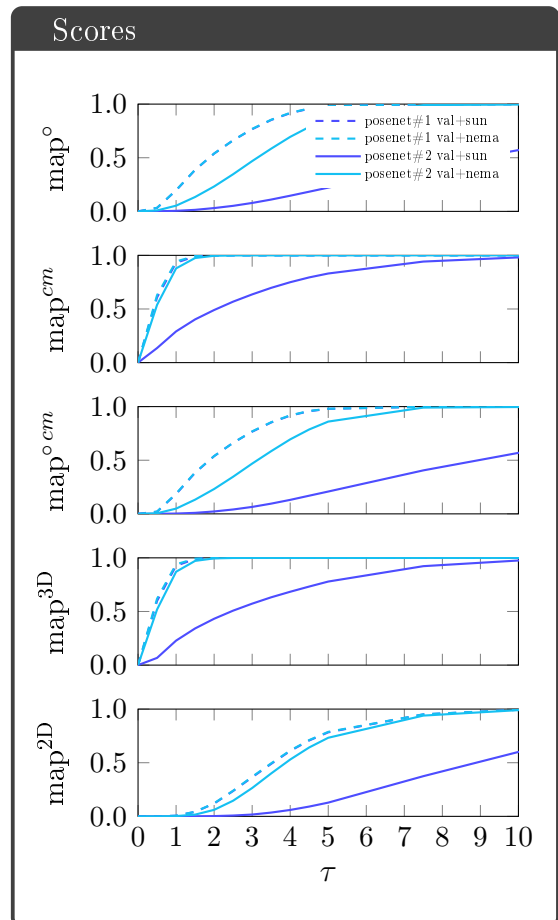
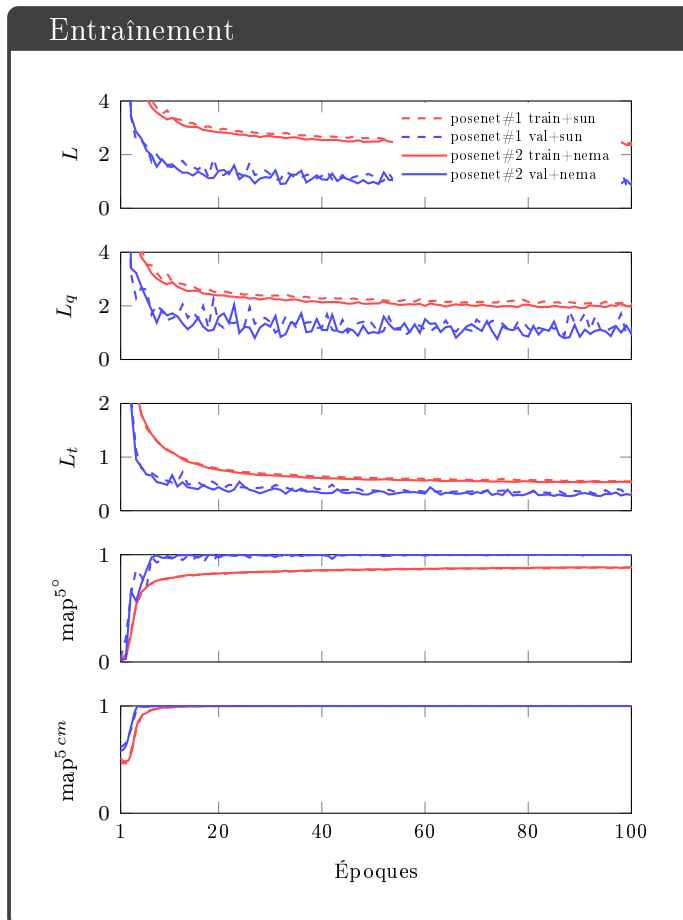
Interprétation

PoseNet apprend bien à prédire la pose du moteur NEMA-17 lorsque les images de SUN sont utilisées comme arrière-plans. Les images d'entraînement présentant des poses acquises plus densément, PoseNet apprend à prédire la pose de façon très précise comparé à LINEMOD, T-LESS ou YCB-VIDÉO.

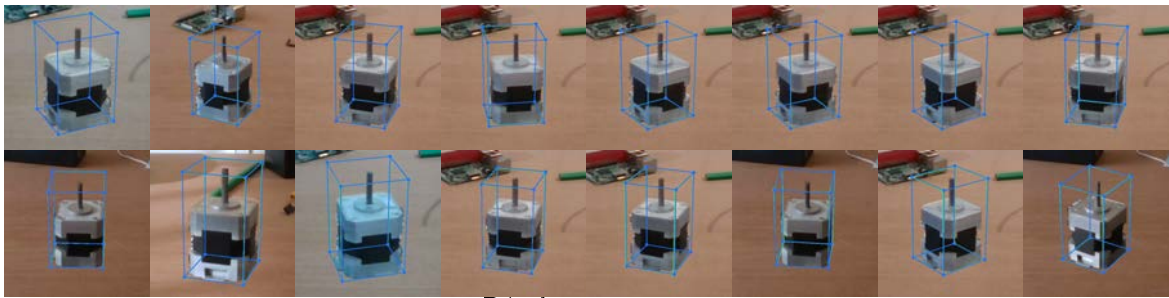
A.30 POSENET / NEMA / #2 (arrière-plans NEMA)

Implémentation

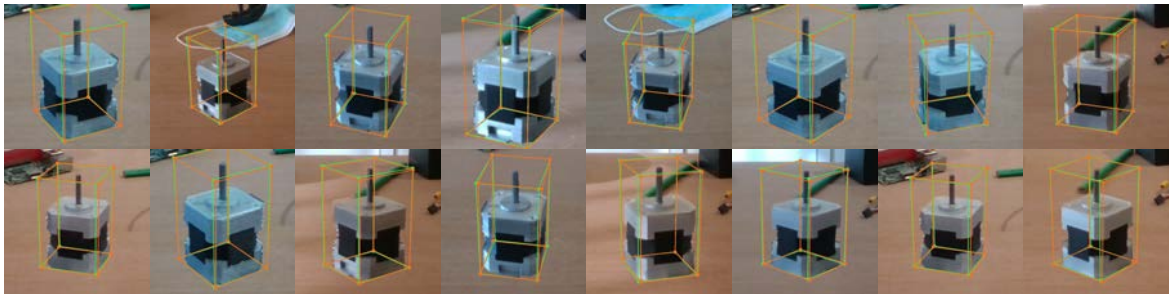
- 🏠 **Modèle :**
 - 🔗 Réseau : PoseNet (GoogLeNet)
 - 📄 Entrée : $227 \times 227 \times 3$
 - 📄 Sortie : $4 + 3 (q + t)$
 - 🔗 Activation : Linéaire
- 🖼️ **Images :**
 - 📁 Jeu de données : NEMA
 - 🔄 Répartition : 132 480 entraînement et 40 320 validation/tests
 - 🍷 Préparation :
 - 📐 Recadrage à 227×227 pixels centré sur l'objet
 - 😊 Remplacement de l'arrière-plan avec l'arrière-plan réel
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînement
- 📉 **Descente de gradient :**
 - 🕒 Fonction de coût : distance euclidienne
 - 📅 Nombre d'époques : 100



Résultats



Résultats corrects



Résultats incorrects

Interprétation

PoseNet apprend bien à prédire la pose du moteur NEMA-17 lorsque les arrière-plans réels sont utilisés. Lorsque nous évaluons cette version de PoseNet sur des images avec les arrière-plans réel les résultats sont moins bons que ceux de la version précédente entraînée avec des images de SUN comme arrière-plans. Cette versions généralise aussi beaucoup moins bien sur des images de test où des images de SUN sont utilisées comme arrière-plans. Utiliser des images de SUN comme arrière-plan est donc préférable pour PoseNet car cela lui permet de mieux généraliser.

A.31 BB8 / NEMA / #1 (arrière-plans SUN)

Implémentation

📦 Modèle :

🔗 Réseau : BB8 (VGG-19)

📏 Sortie : 8×2 (8)

📄 Entrée : $224 \times 224 \times 3$

🔌 Activation : Linéaire

🖼️ Images :

📁 Jeu de données : NEMA

🔄 Répartition : 132 480 entraînement et 40 320 validation/tests

🍹 Préparation :

📐 Recadrage à 227×227 pixels centré sur l'objet

📏 Mise à l'échelle à 224×224

😊 Remplacement de l'arrière-plan avec une image de SUN

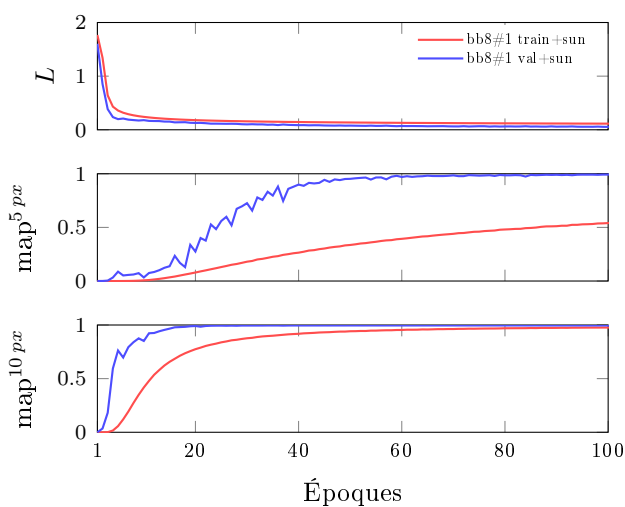
💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînements

📉 Descente de gradient :

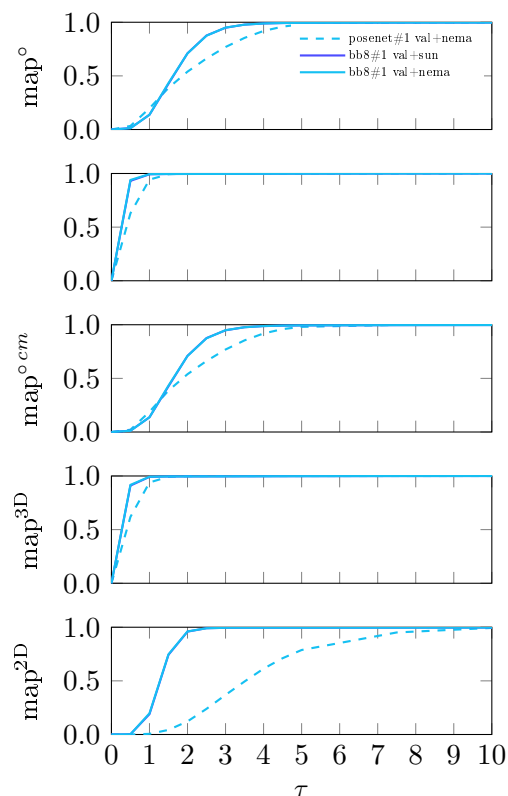
🕒 Fonction de coût : distance euclidienne

📦 Nombre d'époques : 100

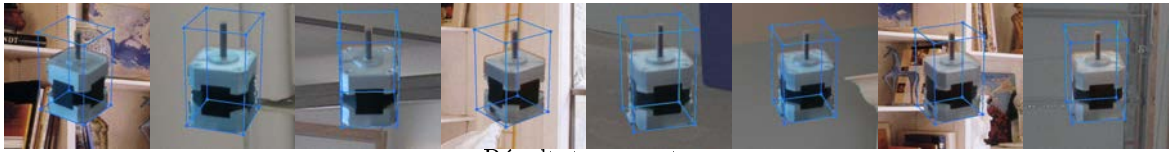
Entraînement



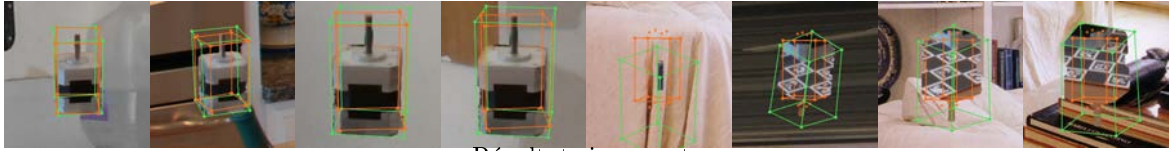
Scores



Résultats



Résultats corrects



Résultats incorrects

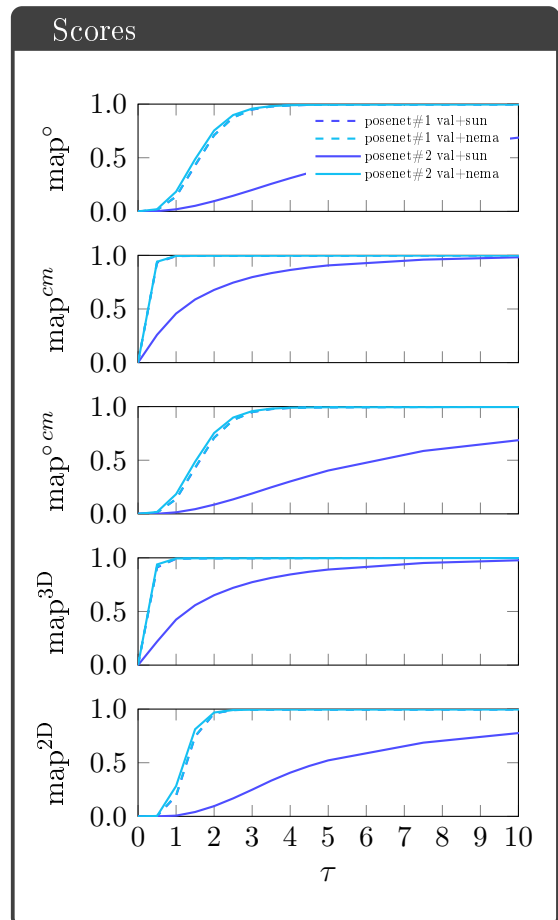
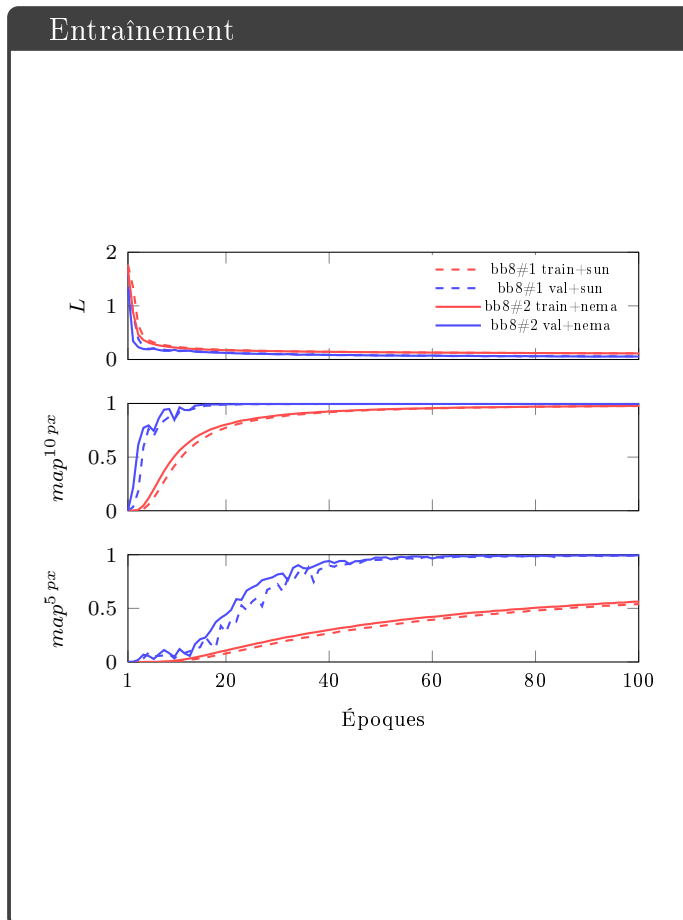
Interprétation

BB8 apprend parfaitement à prédire la pose de l'objet de NEMA avec des arrière-plans tirés d'images de SUN. Les résultats sont bien meilleurs que ceux de PoseNet.

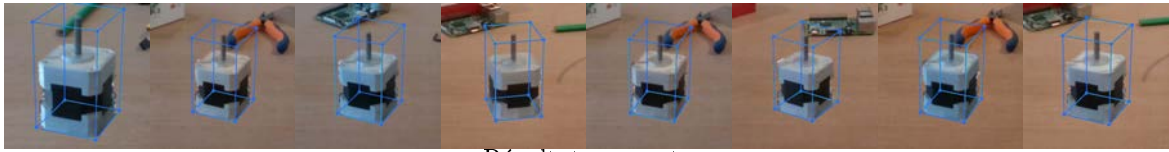
A.32 BB8 / NEMA / #2 (arrière-plans NEMA)

Implémentation

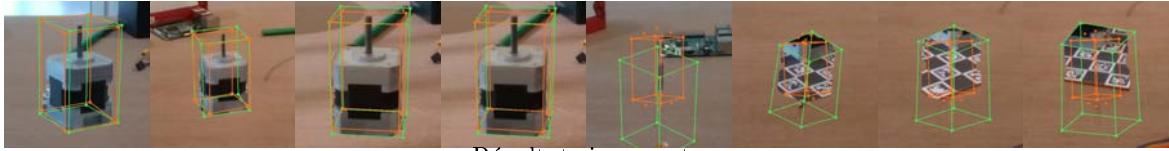
- 📦 Modèle :
 - 🔗 Réseau : BB8 (VGG-19) 📄 Sortie : 8×2 (8)
 - 📄 Entrée : 224×224×3 🔌 Activation : Linéaire
- 📁 Images :
 - 📄 Jeu de données : NEMA
 - 🔄 Répartition : 132 480 entraînement et 40 320 validation/tests
 - 🍹 Préparation :
 - 📐 Recadrage à 227×227 pixels centré sur l'objet
 - 📏 Mise à l'échelle à 224×224
 - 😊 Remplacement de l'arrière-plan avec l'arrière-plan réel
 - 💡 Variations aléatoires de luminosité, contraste, teinte et saturation dans les images d'entraînements
- 📉 Descente de gradient :
 - 🕒 Fonction de coût : distance euclidienne
 - 📦 Nombre d'époques : 100



Résultats



Résultats corrects



Résultats incorrects

Interprétation

BB8 apprend parfaitement à prédire la pose de l'objet de NEMA avec les arrière-plans réels. Les résultats sur des images de test avec les arrière-plans réel sont meilleurs que la version précédente de BB8 entraînée sur des images avec des arrière-plans tirés d'images de SUN. Mais cette version de BB8 ne généralise pas sur des images où les arrière-plans sont issus d'images de SUN. Elle est spécialisée sur les images de NEMA avec les arrière-plans réels.

Bibliographie

- [1] 3BLUE1BROWN. *Quaternions and 3D rotation explained interactively*. 2018. URL : <https://www.youtube.com/watch?v=zjMuIxRvygQ>.
- [2] 3BLUE1BROWN. *Visualizing Quaternions with stereographic projection*. 2018. URL : <https://www.youtube.com/watch?v=d4EgbgTm0Bg>.
- [3] AEROSULPTURE. *Site internet Aerosculpture*. 2019. URL : <https://aerosculpture.com/>.
- [4] Herbert BAY et al. « Speeded-up robust features (SURF) ». In : *Computer Vision and Image Understanding (CVIU)* 110.3 (juin 2008), p. 346-359. URL : <https://www.sciencedirect.com/science/article/abs/pii/S1077314207001555>.
- [5] Serge BELONGIE, Jitendra MALIK et Jan PUZICHA. « Matching shapes ». In : *International Conference on Computer Vision (ICCV)*. Vancouver, Colombie-Britannique, Canada, juill. 2001. URL : <https://ieeexplore.ieee.org/document/937552?arnumber=937552>.
- [6] P.J. BESL et Neil D. MCKAY. « A method for registration of 3-D shapes ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence (ITPM)* 14.2 (fév. 1992), p. 239-256. URL : <https://ieeexplore.ieee.org/document/121791>.
- [7] Alexey BOCHKOVSKIY, Chien-Yao WANG et Liao Hong-Yuan MARK. « YOLOv4 : Optimal speed and accuracy of object detection ». In : *arXiv, Computing Research Repository (CoRR)* abs/2004.10934 (avr. 2020), p. 1-17. URL : <https://arxiv.org/abs/2004.10934>.
- [8] Navaneeth BODLA et al. « Soft-NMS : Improving object detection with one line of code ». In : *International Conference on Computer Vision (ICCV)*. Venise, Italie, oct. 2017. URL : https://openaccess.thecvf.com/content_iccv_2017/html/Bodla_Soft-NMS_-_Improving_ICCV_2017_paper.html.
- [9] Eric BRACHMANN et al. « Learning 6D object pose estimation using 3D object coordinates ». In : *European Conference on Computer Vision (ECCV)*. Zürich, Suisse, sept. 2014. URL : https://link.springer.com/chapter/10.1007/978-3-319-10605-2_35.
- [10] Eric BRACHMANN et al. « Uncertainty-Driven 6D pose estimation of objects and scenes from a single RGB image ». In : *Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Néveda, États-Unis, juin 2016. URL : https://openaccess.thecvf.com/content_cvpr_2016/html/Brachmann_Uncertainty-Driven_6D_Pose_CVPR_2016_paper.html.
- [11] Matthew BRETT. *Bibliothèque Python transforms3d*. 2017. URL : <https://pypi.org/project/transforms3d/>.

- [12] Phong BUI TUONG. « Illumination for computer generated pictures ». In : *Association for Computing Machinery (ACM)* 18.6 (juin 1975), p. 311-317. URL : <http://doi.acm.org/10.1145/360825.360839>.
- [13] Hongping CAI, Tomáš WERNER et Jiří MATAS. « Fast detection of multiple textureless 3-D objects ». In : *Computer Vision Systems (ICVS)*. Saint Pétersbourg, Russia, juill. 2013. URL : https://link.springer.com/chapter/10.1007/978-3-642-39402-7_11.
- [14] Berk CALLI et al. « Benchmarking in manipulation research : Using the Yale-CMU-Berkeley object and model set ». In : *IEEE Robotics Automation Magazine* 22.3 (sept. 2015), p. 36-52. URL : <https://ieeexplore.ieee.org/document/7254318>.
- [15] Berk CALLI et al. « The YCB object and model set : Towards common benchmarks for manipulation research ». In : *International Conference on Advanced Robotics (ICAR)*. Istanbul, Turquie, juill. 2015. URL : <https://ieeexplore.ieee.org/document/7251504>.
- [16] Berk CALLI et al. « Yale-CMU-Berkeley dataset for robotic manipulation research ». In : *The International Journal of Robotics Research* 36.3 (avr. 2017), p. 261-268. URL : <https://journals.sagepub.com/doi/10.1177/0278364917700714>.
- [17] François CHOLLET. « Xception : Deep learning with depthwise separable convolutions ». In : *Computer Vision and Pattern Recognition (CVPR)*. Honolulu, Hawaï, États-Unis, juill. 2017. URL : https://openaccess.thecvf.com/content_cvpr_2017/html/Chollet_Xception_Deep_Learning_CVPR_2017_paper.html.
- [18] Francois CHOLLET et al. *Keras*. 2015. URL : <https://github.com/fchollet/keras>.
- [19] D. COMANICIU et P. MEER. « Mean shift : A robust approach toward feature space analysis ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 24.5 (mai 2002), p. 603-619. URL : <https://ieeexplore.ieee.org/document/1000236>.
- [20] Runmin CONG et al. « Review of visual saliency detection with comprehensive information ». In : *IEEE Transactions on Circuits and Systems for Video Technology (CSVT)* 29.10 (sept. 2019), p. 2941-2959.
- [21] Alberto CRIVELLARO et al. « A novel representation of parts for accurate 3D object detection and tracking in monocular images ». In : *International Conference on Computer Vision (ICCV)*. Las Condes, Chili, déc. 2015. URL : https://openaccess.thecvf.com/content_iccv_2015/html/Crivellaro_A_Novel_Representation_ICCV_2015_paper.html.
- [22] James CROWLEY et Arthur SANDERSON. « Multiple resolution representation and probabilistic matching of 2-D gray-scale shape ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 9.1 (déc. 1987), p. 113-121. URL : <https://ieeexplore.ieee.org/document/4767876>.
- [23] Jeffrey DEAN et al. « Large scale distributed deep networks ». In : *Neural Information Processing Systems (NeurIPS)*. Stateline, Nevada, États-Unis, déc. 2012. URL : <https://papers.nips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html>.
- [24] Thomas DEAN et al. « Fast, accurate detection of 100,000 object classes on a single machine ». In : *Computer Vision and Pattern Recognition (CVPR)*. Portland, Orégon, États-Unis, juin 2013. URL : https://openaccess.thecvf.com/content_cvpr_2013/html/Dean_Fast_Accurate_Detection_2013_CVPR_paper.html.

- [25] Jia DENG et al. « ImageNet : A large-scale hierarchical image database ». In : *Computer Vision and Pattern Recognition (CVPR)*. Miami, Floride, États-Unis, 2009. URL : <https://ieeexplore.ieee.org/document/5206848>.
- [26] James DIEBEL. « Representing attitude : Euler angles, unit quaternions, and rotation vectors ». In : *Matrix* 58.15 (jan. 2006), p. 1-35.
- [27] Alexey DOSOVITSKIY et al. « An image is worth 16x16 words : transformers for image recognition at scale ». In : *International Conference on Learning Representations (ICLR)*. Online, mai 2021. URL : <https://arxiv.org/abs/2010.11929>.
- [28] Andreas DOUMANOGLOU et al. « Recovering 6D object pose and predicting next-best-view in the crowd ». In : *Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Nevada, États-Unis, juin 2016. URL : https://openaccess.thecvf.com/content_cvpr_2016/html/Doumanoglou_Recovering_6D_Object_CVPR_2016_paper.html.
- [29] Bertram DROST et al. « Introducing MVTec ITODD — A Dataset for 3D Object Recognition in Industry ». In : *International Conference on Computer Vision (ICCV)*. Venise, Italie, oct. 2017. URL : https://openaccess.thecvf.com/content_ICCV_2017_workshops/w31/html/Drost_Introducing_MVTec_ITODD_ICCV_2017_paper.html.
- [30] Richard O. DUDA et Peter E. HART. « Use of the Hough rransformation to detect lines and curves in pictures ». In : *Association for Computing Machinery (ACM)* 15.1 (jan. 1972), p. 11-15. URL : <https://dl.acm.org/doi/10.1145/361237.361242>.
- [31] Debidatta DWIBEDI, Ishan MISRA et Martial HEBERT. « Cut, paste and learn : Surprisingly easy synthesis for instance detection ». In : *International Conference on Computer Vision (ICCV)*. Venise, Italie, oct. 2017. URL : <https://ieeexplore.ieee.org/document/8237408/>.
- [32] EUROSONO. *Site internet Eurosono*. 2019. URL : <https://www.eurosono.com/>.
- [33] Mark EVERINGHAM et al. « The pascal visual object classes (VOC) challenge ». In : *International Journal of Computer Vision* 88.2 (juin 2010), p. 303-338. URL : <https://link.springer.com/article/10.1007/s11263-009-0275-4>.
- [34] Pedro FELZENSZWALB et al. « Object detection with discriminatively trained part-based models ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence (ITPM)* 32.9 (sept. 2010), p. 1627-1645. URL : <https://ieeexplore.ieee.org/document/5255236>.
- [35] Pedro F. FELZENSZWALB et Daniel P. HUTTENLOCHER. « Efficient Graph-Based Image Segmentation ». In : *International Journal of Computer Vision* 59.2 (sept. 2004), p. 167-181. URL : <https://link.springer.com/article/10.1023/B:VISI.0000022288.19776.77>.
- [36] Martin FISCHLER et Robert BOLLES. « Random sample consensus : A paradigm for model fitting with applications to image analysis and automated cartography ». In : *Association for Computing Machinery (ACM)* 24.6 (juin 1981), p. 381-395. URL : <https://dl.acm.org/doi/10.1145/358669.358692>.
- [37] William FREEMAN et Michal ROTH. « Orientation histograms for hand gesture recognition ». In : *Automatic Face and Gesture Recognition*. Zürich, Suisse, juin 1994. URL : <https://www.merl.com/publications/TR94-03>.

- [38] Guillermo GALLEGRO et Anthony YEZZI. « A compact formula for the derivative of a 3-D rotation in exponential coordinates ». In : *Journal of Mathematical Imaging and Vision* 51.3 (mars 2015), p. 378-384. URL : <https://link.springer.com/article/10.1007/s10851-014-0528-x>.
- [39] Sergio GARRIDO-JURADO et al. « Automatic generation and detection of highly reliable fiducial markers under occlusion ». In : *Association for Computing Machinery (ACM)* 47.6 (juin 2014), p. 2280-2292. URL : <https://dl.acm.org/doi/10.1016/j.patcog.2014.01.005>.
- [40] GARSENTX. *Pyramide holographique sur Amazon*. 2021. URL : https://www.amazon.fr/Projecteur-Holographique-Hologramme-dAffichage-Expositions/dp/B08797WY2P/ref=sr_1_36.
- [41] Christophe GAZEAU. *Ludification digitale chez Mc Donalds*. 2013. URL : <https://christophegazeau.wordpress.com/2013/10/15/ludification-digitale-chez-mc-donalds/>.
- [42] Andreas GEIGER, Philip LENZ et Raquel URTASUN. « Are we ready for autonomous driving? The KITTI vision benchmark suite ». In : *Computer Vision and Pattern Recognition (CVPR)*. Providence, Rhode Island, États-Unis, juin 2012. URL : <https://ieeexplore.ieee.org/document/6248074>.
- [43] Aritra GHOSH, Himanshu KUMAR et P. S. SASTRY. « Robust loss functions under label noise for deep neural networks ». In : *Conference on Artificial Intelligence (AAAI)*. AAAI'17. San Francisco, Californie, États-Unis, fév. 2017.
- [44] Ross GIRSHICK. « Fast R-CNN ». In : *International Conference on Computer Vision (ICCV)*. Las Condes, Chili, déc. 2015. URL : https://openaccess.thecvf.com/content_iccv_2015/html/Girshick_Fast_R-CNN_ICCV_2015_paper.html.
- [45] Ross GIRSHICK et al. « Deformable part models are convolutional neural networks ». In : *Computer Vision and Pattern Recognition (CVPR)*. Boston, États-Unis, juin 2015. URL : https://openaccess.thecvf.com/content_cvpr_2015/html/Girshick_Deformable_Part_Models_2015_CVPR_paper.html.
- [46] Ross GIRSHICK et al. « Rich feature hierarchies for accurate object detection and semantic segmentation ». In : *Computer Vision and Pattern Recognition (CVPR)*. Columbus, Ohio, États-Unis, juin 2014. URL : https://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html.
- [47] Jonathan HAREL, Christof KOCH et Pietro PERONA. « Graph-based visual saliency ». In : *Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Colombie-Britannique, Canada, déc. 2006. URL : <https://papers.nips.cc/paper/2006/hash/4db0f8b0fc895da263fd77fc8aecabe4-Abstract.html>.
- [48] Kaiming HE et al. « Deep residual learning for image recognition ». In : *Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Nevada, États-Unis, juin 2016. URL : https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.
- [49] Stefan HINTERSTOISSER et al. « Gradient response maps for real-time detection of textureless objects ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence (ITPM)* 34.5 (oct. 2012), p. 876-888. URL : <https://ieeexplore.ieee.org/document/6042881>.

- [50] Stefan HINTERSTOISSER et al. « Model-based training, detection and pose estimation of texture-less 3D objects in heavily cluttered Scenes ». In : *Asian Conference Computer Vision (ACCV)*. Daejeon, Korea, nov. 2013. URL : https://link.springer.com/chapter/10.1007/978-3-642-37331-2_42.
- [51] Stefan HINTERSTOISSER et al. « Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes ». In : *International Conference on Computer Vision (ICCV)*. Barcelone, Espagne, nov. 2011. URL : <https://ieeexplore.ieee.org/document/6126326>.
- [52] Tin Kam HO. « Random decision forests ». In : *International Conference on Document Analysis and Recognition*. Montréal, Québec, Canada, août 1995. URL : <https://ieeexplore.ieee.org/document/598994>.
- [53] Tomáš HODANĚ et al. « BOP challenge 2020 on 6D object localization ». In : *European Conference on Computer Vision (ECCV 2020)*. Online, août 2020. URL : <https://arxiv.org/abs/2009.07378>.
- [54] Tomáš HODANĚ et al. « BOP : Benchmark for 6D object pose estimation ». In : *European Conference on Computer Vision (ECCV 2018)*. Munich, Allemagne, sept. 2018. URL : <https://arxiv.org/abs/1808.08319>.
- [55] Tomáš HODANĚ et al. « Detection and fine 3D pose estimation of texture-less objects in RGB-D images ». In : *Intelligent Robots and Systems (IROS)*. Hambourg, Allemagne, sept. 2015. URL : <https://ieeexplore.ieee.org/document/7354005>.
- [56] Tomáš HODANĚ et al. « T-LESS : An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects ». In : *Winter Conference on Applications of Computer Vision (WACV)*. Santa Rosa, Californie, États-Unis, mars 2017. URL : <https://ieeexplore.ieee.org/document/7926686>.
- [57] Paul HOUGH. « Methods and means for recognizing complex patterns ». Déc. 1962.
- [58] Andrew G. HOWARD et al. « MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications ». In : *arXiv, Computing Research Repository (CoRR)* abs/1704.04861 (avr. 2017). URL : <https://arxiv.org/abs/1704.04861>.
- [59] IKEA. *Application mobile IKEA Place*. 2017. URL : <https://www.ikea.com/fr/fr/customer-service/mobile-apps/>.
- [60] INTEL. *Cameras Intel RealSense*. 2015. URL : <https://www.intel.fr/content/www/fr/fr/architecture-and-technology/realsense-overview.html>.
- [61] Sergey IOFFE et Christian SZEGEDY. « Batch Normalization : Accelerating deep network training by reducing internal covariate shift ». In : *International Conference on Machine Learning (ICML)*. Lille, France, juill. 2015. URL : <https://arxiv.org/abs/1502.03167>.
- [62] Roman KASKMAN et al. « HomebrewedDB : RGB-D Dataset for 6D pose estimation of 3D Objects ». In : *International Conference on Computer Vision (ICCVW)*. Séoul, Corée du Sud, oct. 2019. URL : <https://www.computer.org/csdl/proceedings-article/iccvw/2019/502300c767/1i5mlBAKkGk>.
- [63] Alex KENDALL, Matthew GRIMES et Roberto CIPOLLA. « PoseNet : A convolutional networks for real-time 6-DOF camera relocalization ». In : *International Conference on Computer Vision (ICCV)*. Santiago, Chili, déc. 2015. URL : https://openaccess.thecvf.com/content_iccv_2015/html/Kendall_PoseNet_A_Convolutional_ICCV_2015_paper.html.

- [64] Leonid KESELMAN et al. « Intel RealSense stereoscopic depth cameras ». In : *Computer Vision and Pattern Recognition (CVPR)*. Honolulu, Hawaï, États-Unis, juill. 2017. URL : https://openaccess.thecvf.com/content_cvpr_2017_workshops/w15/html/Keselman_Intel_RealSense_Stereoscopic_CVPR_2017_paper.html.
- [65] Diederik KINGMA et Jimmy BA. « Adam : A method for stochastic optimization ». In : *International Conference on Learning Representations (ICLR)*. San Diego, Californie, États-Unis, 2015. URL : <http://arxiv.org/abs/1412.6980>.
- [66] John F. KOLEN et Stefan C. KREMER. « Gradient Flow in Recurrent Nets : The Difficulty of Learning LongTerm Dependencies ». In : *A Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 2001, p. 237-243. DOI : 10.1109/9780470544037.ch14.
- [67] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E. HINTON. « ImageNet classification with deep convolutional neural networks ». In : *Association for Computing Machinery (ACM) 60.6* (mai 2017), p. 84-90. URL : <https://dl.acm.org/doi/10.1145/3065386>.
- [68] Jack B. KUIPERS. *Quaternions and rotation sequences : a primer with applications to orbits, aerospace, and virtual reality*. Princeton Univerty Press, sept. 1999. URL : <https://press.princeton.edu/books/paperback/9780691102986/quaternions-and-rotation-sequences>.
- [69] Yan LECUN et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE 86.11* (nov. 1998), p. 2278-2324. URL : <https://ieeexplore.ieee.org/document/726791>.
- [70] LEGO. *Thème Hidden-Side : jouets en réalité augmentée*. 2019. URL : <https://www.lego.com/fr-fr/themes/hidden-side/ar-games>.
- [71] Vincent LEPETIT, Francesc MORENO-NOGUER et Pascal FUA. « EPnP : An accurate O(n) solution to the PnP problem ». In : *International Journal of Computer Vision (IJCV) 81.2* (juill. 2008), p. 3828-3836. URL : <https://link.springer.com/article/10.1007/s11263-008-0152-6>.
- [72] Yi LI et al. « DeepIM : Deep iterative matching for 6D pose estimation ». In : *International Journal of Computer Vision (IJCV) 128.3* (mars 2018), p. 657-678. URL : <https://link.springer.com/article/10.1007/s11263-019-01250-9>.
- [73] Ming LIANG et Xiaolin HU. « Recurrent Convolutional Neural Network for Object Recognition ». In : *Computer Vision and Pattern Recognition (CVPR)*. Boston, Massachusetts, États-Unis, juin 2015. URL : https://openaccess.thecvf.com/content_cvpr_2015/html/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.html.
- [74] Tsung-Yi LIN et al. « Microsoft COCO : Common objects in context ». In : *European Conference on Computer Vision (ECCV)*. Zürich, Suisse, sept. 2014. URL : https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48.
- [75] David LOWE. « Object recognition from local scale-invariant features ». In : *International Conference on Computer Vision (ICCV)*. Vancouver, Colombie-Britannique, Canada, juill. 2001. URL : <https://ieeexplore.ieee.org/document/790410>.
- [76] Chien-Ping LU, Gregory HAGER et Eric MJOLSNESS. « Fast and globally convergent pose estimation from video images ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence 22.6* (juin 2000), p. 610-622. URL : <https://ieeexplore.ieee.org/document/862199>.

- [77] MICROSOFT. *Microsoft Hololens 2*. 2021. URL : <https://www.microsoft.com/fr-fr/hololens>.
- [78] MICROSOFT. *Présentation de Dynamics 365 Guides*. 2016. URL : <https://docs.microsoft.com/fr-fr/dynamics365/mixed-reality/guides/>.
- [79] Paul MILGRAM et Fumio KISHINO. « A Taxonomy of Mixed Reality Visual Displays ». In : *IEICE Transactions on Information and Systems* E77-D.12 (déc. 1994), p. 1321-1329. URL : https://cs.gmu.edu/~zduric/cs499/Readings/r76JBo-Milgram-IEICE_1994.pdf.
- [80] George A. MILLER et al. « Introduction to WordNet : An on-line lexical database ». In : *International Journal of Lexicography (IJL)* 3.4 (jan. 1991), p. 235-244. URL : <https://academic.oup.com/ijl/article-abstract/3/4/235/923280>.
- [81] Kevin Patrick MURPHY. *Machine learning : a probabilistic perspective*. 1^{re} éd. MIT press, août 2012. URL : <https://probml.github.io/pml-book/book0.html>.
- [82] NEMA. *NEMA ICS 16, Industrial control and systems, motion/position, control motors, controls and feedback Devices*. Rapp. tech. National Electrical Manufacturers Association, 2001. URL : <https://www.nema.org/standards/view/motion-position-control-motors-controls-and-feedback-devices>.
- [83] F. H. NETTER et al. *Atlas d'anatomie humaine*. 5^e éd. Elsevier Health Sciences, 2012. URL : https://books.google.fr/books?id=V%5C_QoDwAAQBAJ.
- [84] Fondation OPENCV. *Open computer vision library*. 2020. URL : <https://opencv.org/>.
- [85] Stanley OSHER et Ronald FEDKIW. « Signed Distance Functions ». In : *Level Set Methods and Dynamic Implicit Surfaces*. New York, NY : Springer New York, 2003, p. 17-22. ISBN : 978-0-387-22746-7. DOI : 10.1007/0-387-22746-6_2. URL : https://doi.org/10.1007/0-387-22746-6_2.
- [86] Sida PENG et al. « PVNet : Pixel-wise voting network for 6DoF pose estimation ». In : *Computer Vision and Pattern Recognition (CVPR)*. Long Beach, Californie, États-Unis, juin 2019. URL : https://openaccess.thecvf.com/content_CVPR_2019/html/Peng_PVNet_Pixel-Wise_Voting_Network_for_6DoF_Pose_Estimation_CVPR_2019_paper.html.
- [87] Philippe PÉREZ DE SAN ROMAN et al. « Limitations of metric loss for the estimation of joint translation and rotation ». In : *Computer Vision Theory and Applications (VISAPP)*. Prague, République-Tchèque, fév. 2019. URL : <https://www.scitepress.org/Link.aspx?doi=10.5220/0007525005900597>.
- [88] Philippe PÉREZ DE SAN ROMAN et al. « Saliency driven object recognition in egocentric videos with deep CNN : toward application in assistance to neuroprostheses ». In : *Computer Vision and Image Understanding (CVIU)* 164 (nov. 2017), p. 82-91. URL : <https://www.sciencedirect.com/science/article/abs/pii/S1077314217300462>.
- [89] Matti PIETIKINEN et al. *Computer Vision Using Local Binary Patterns*. 1^{re} éd. Springer Publishing Company, 2011. URL : <https://link.springer.com/book/10.1007/978-0-85729-748-8>.

- [90] Giorgia PITTERI et al. « 3D object detection and pose estimation of unseen objects in color images with local surface embeddings ». In : *Asian Conference on Computer Vision (ACCV)*. Kyoto Virtuel, Japon Virtuel, nov. 2021. URL : https://openaccess.thecvf.com/content/ACCV2020/html/Pitteri_3D_Object_Detection_and_Pose_Estimation_of_Unseen_Objects_in_ACCV_2020_paper.html.
- [91] 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 : *International Conference on Computer Vision (ICCV)*. Venise, Italie, oct. 2017. URL : <https://arxiv.org/abs/1703.10896>.
- [92] Joseph REDMON et Ali FARHADI. « YOLO9000 : Better, faster, stronger ». In : *Computer Vision and Pattern Recognition (CVPR)*. Honolulu, Hawaï, États-Unis, juill. 2017. URL : https://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html.
- [93] Joseph REDMON et Ali FARHADI. « YOLOv3 : An incremental improvement ». In : *arXiv, Computing Research Repository (CoRR)* abs/1804.02767 (avr. 2018), p. 1-6. URL : <https://arxiv.org/abs/1804.02767>.
- [94] Joseph REDMON et al. « You Only Look Once : Unified, Real-Time Object Detection ». In : *Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Néveda, États-Unis, juin 2016. URL : https://openaccess.thecvf.com/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.
- [95] Shaoqing REN et al. « Faster R-CNN : Towards real-time object detection with region proposal networks ». In : *Neural Information Processing Systems (NeurIPS)*. Montréal, Canada, déc. 2015. URL : <https://papers.nips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html>.
- [96] Colin RENNIE et al. « A Dataset for Improved RGBD-based object detection and pose estimation for warehouse pick-and-place ». In : *IEEE Robotics and Automation Letters* 1.2 (sept. 2015), p. 1179-1185. URL : <https://ieeexplore.ieee.org/document/7416002>.
- [97] Olaf RONNEBERGER, Philipp FISCHER et Thomas BROX. « U-Net : Convolutional networks for biomedical image segmentation ». In : *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Munich, Allemagne, oct. 2015. URL : https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28.
- [98] Frank ROSENBLATT. *The perceptron — A perceiving and recognizing automaton*. Rapp. tech. Cornell Aeronautical Laboratory, 1957.
- [99] Randi J. ROST et al. *OpenGL shading language*. Addison-Wesley, juill. 2009. URL : <https://www.amazon.fr/OpenGL-Shading-Language-Randi-Rost/dp/0321637631>.
- [100] Nicolas ROUGIER. *Python and OpenGL for scientific visualization*. LaBRI, 2018.
- [101] Olga RUSSAKOVSKY et al. « ImageNet large scale visual recognition challenge ». In : *International Journal of Computer Vision (IJCV)* 115.3 (2015), p. 211-252. DOI : 10.1007/s11263-015-0816-y. URL : <https://link.springer.com/article/10.1007/s11263-015-0816-y>.
- [102] Jorge SANCHEZ et al. « Image classification with the Fisher vector : theory and practice ». In : *International Journal of Computer Vision (IJCV)* 105.3 (déc. 2013), p. 222-245. URL : <https://link.springer.com/article/10.1007/s11263-013-0636-x>.

- [103] Grant SANDERSON et Ben EATER. *Visualizing quaternions*. 2018. URL : <https://eater.net/quaternions>.
- [104] Mark SANDLER et al. « Inverted residuals and linear bottlenecks : mobile networks for classification, detection and segmentation ». In : *Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, Utah, États-Unis, juin 2018. URL : https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.
- [105] Bernhard SCHÖLKOPF et SAlexander J. SMOLA. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, 2002. URL : <https://mitpress.mit.edu/books/learning-kernels>.
- [106] Dave SHREINER et al. *OpenGL programming guide : The official guide to learning OpenGL, version 4.3*. 8^e éd. Addison-Wesley, mars 2013. URL : <https://www.amazon.fr/OpenGL-Programming-Guide-Official-Learning/dp/0321773039>.
- [107] Laurent SIFRE. « Rigid-motion scattering for image classification ». Thèse de doct. Ecole Polytechnique, CMAP, 2014.
- [108] Karen SIMONYAN et Andrew ZISSERMAN. « Very deep convolutional networks for large-scale image recognition ». In : *International Conference on Learning Representations (ICLR)*. San Diego, Californie, États-Unis, mai 2015. URL : <https://arxiv.org/abs/1409.1556>.
- [109] Chen SONG, Jiaru SONG et Qixing HUANG. « HybridPose : 6D object pose estimation under hybrid representations ». In : *Computer Vision and Pattern Recognition (CVPR)*. Online, juin 2020. URL : https://openaccess.thecvf.com/content_CVPR_2020/html/Song_HybridPose_6D_Object_Pose_Estimation_Under_Hybrid_Representations_CVPR_2020_paper.html.
- [110] Ilya SUTSKEVER et al. « On the importance of initialization and momentum in deep learning ». In : *International Conference on Machine Learning (ICML)*. Atlanta, Georgia, USA, juin 2013. URL : <https://proceedings.mlr.press/v28/sutskever13.html>.
- [111] Christian SZEGEDY et al. « Going deeper with convolutions ». In : *Computer Vision and Pattern Recognition (CVPR)*. Boston, États-Unis, juin 2015. URL : https://openaccess.thecvf.com/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html.
- [112] Christian SZEGEDY et al. « Rethinking the inception architecture for computer vision ». In : *Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, Nevada, États-Unis, juin 2016.
- [113] Mingxing TAN et Quoc LE. « EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks ». In : *International Conference on Machine Learning (ICML)*. Long Beach, Californie, États-Unis, juin 2019. URL : <https://arxiv.org/abs/1905.11946>.
- [114] Alykhan TEJANI et al. « Latent-class Hough forests for 3D object detection and pose estimation ». In : *European Conference on Computer Vision (ECCV)*. Zürich, Suisse, sept. 2014. URL : https://link.springer.com/chapter/10.1007/978-3-319-10599-4_30.

- [115] Bugra TEKIN, Sudipta SINHA et Pascal FUA. « Real-time seamless single shot 6D object pose prediction ». In : *Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, Utah, États-Unis, juin 2018. URL : https://openaccess.thecvf.com/content_cvpr_2018/html/Tekin_Real-Time_Seamless_Single_CVPR_2018_paper.html.
- [116] Jonathan TREMBLAY, Thang TO et Stan BIRCHFIELD. « Falling Things : A synthetic dataset for 3D object detection and pose estimation ». In : *Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, Utah, États-Unis, juin 2018. URL : https://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w40/Tremblay_Falling_Things_A_CVPR_2018_paper.pdf.
- [117] Jonathan TREMBLAY et al. « Deep object pose estimation for semantic robotic grasping of household objects ». In : *Conference on Robot Learning (CoRL)*. Zürich, Suisse, oct. 2018. URL : <https://arxiv.org/abs/1809.10790>.
- [118] J. R. UIJLINGS et al. « Selective search for object recognition ». In : *International Journal of Computer Vision* 104.2 (sept. 2013), p. 154-171.
- [119] VANA. *Projecteur Holographique 3D sur Amazon*. 2021. URL : https://www.amazon.fr/VANA-Hologramme-holographique-Compatible-Smartphone/dp/B01N5YCMAS/ref=sr_1_3.
- [120] Vincent VANHOUCHE. *Learning visual representations at scale*. Rapp. tech. GOOGLE, avr. 2014. URL : https://www.google.com/url?q=https%3A%2F%2Fsites.google.com%2Fa%2Fvanhoucke.com%2Fvincent%2Fpublications%2Fvanhoucke-iclr14.pdf&sa=D&sntz=1&usg=AFQjCNE3NV-gjz4K7ka1_up1u0QHpc8ceg.
- [121] Vladimir N. VAPNIK. *Statistical learning theory*. Wiley-Interscience, sept. 1998. URL : <https://www.wiley.com/en-us/Statistical+Learning+Theory-p-9780471030034>.
- [122] Mojo VISION. *Mojo Lens*. 2019. URL : <https://www.mojo.vision/mojo-lens>.
- [123] Fondation WIKIMEDIA. *Matrice de rotation*. 2020. URL : https://fr.wikipedia.org/wiki/Matrice_de_rotation.
- [124] Fondation WIKIMEDIA. *Ombrage de Phong*. 1975. URL : https://fr.wikipedia.org/wiki/Ombrage_de_Phong.
- [125] Fondation WIKIMEDIA. *Quaternions et rotation dans l'espace*. 2014. URL : https://fr.wikipedia.org/wiki/Quaternions_et_rotation_dans_l%27espace.
- [126] Yu XIANG et al. « PoseCNN : A convolutional neural network for 6D object pose estimation in cluttered scenes ». In : *arXiv, Computing Research Repository (CoRR)* abs/1711.00199 (nov. 2017), p. 1-10. URL : <http://arxiv.org/abs/1711.00199>.
- [127] Jianxiong XIAO et al. « SUN Database : Exploring a large collection of scene categories ». In : *International Journal of Computer Vision* 119.1 (août 2016), p. 3-22. URL : <https://link.springer.com/article/10.1007/s11263-014-0748-y>.
- [128] Jianxiong XIAO et al. « SUN database : Large-scale scene recognition from abbey to zoo ». In : *Computer Vision and Pattern Recognition*. San Francisco, Californie, États-Unis, juin 2010. URL : <https://ieeexplore.ieee.org/document/5539970>.
- [129] Sergey ZAKHAROV, Ivan SHUGUROV et Slobodan ILIC. « DPOD : Dense 6D pose object detector in RGB images ». In : *International Conference on Computer Vision (ICCV)*. Séoul, Corée du Sud, oct. 2019. URL : https://openaccess.thecvf.com/content_ICCV_2019/html/Zakharov_DPOD_6D_Pose_Object_Detector_and_Refiner_ICCV_2019_paper.html.

- [130] Ray ZHANG et al. *PoseNet++ : A CNN framework for online pose regression and robot re-localization*. Rapp. tech. University of Michigan, 2018. URL : <https://posenet-mobile-robot.github.io>.
- [131] Zhengyou ZHANG. « Microsoft Kinect sensor and its effect ». In : *IEEE MultiMedia (IEMM)* 19.2 (fév. 2012), p. 4-10. URL : <https://ieeexplore.ieee.org/abstract/document/6190806>.
- [132] Zhilu ZHANG et Mert SABUNCU. « Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels ». In : *Neural Information Processing Systems (NeurIPS)*. Montréal, Canada, déc. 2018. URL : <https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html>.
- [133] Mu ZHU. *Recall, precision and average precision*. Rapp. tech. University of Waterloo, sept. 2004.