Réseaux de neurones convolutionnels profonds pour la détection de petits véhicules en imagerie aérienne

Jean Ogier Du Terrail

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Caen Normandie

Réseaux de Neurones Convolutionnels Profonds pour la détection de petits véhicules en Imagerie Aérienne.

Présentée et soutenue par
Jean OGIER DU TERRAIL

Thèse soutenue publiquement le 20/12/2018 devant le jury composé de		
Mme ELISA FROMONT	Professeur des universités, IRISA/INRIA Rennes	Rapporteur du jury
M. SEBASTIEN LEFEVRE	Professeur des universités, Université Bretagne Sud Vannes	Rapporteur du jury
M. MATTHIEU CORD	Professeur des universités, Université Paris 6 Pierre et Marie Curie	Président du jury
M. FARID OUDYI	Expert, Safran	Membre du jury
M. FREDERIC JURIE	Professeur des universités, Université Caen Normandie	Directeur de thèse

Thèse dirigée par FREDERIC JURIE, Groupe de recherche en informatique, image, automatique et instrumentation

UNIVERSITÉ CAEN NORMANDIE

MIIS

GREYC

Table des matières

Table des figures

Liste des tableaux

Notations et Abréviations

Abréviation	Signification
ACP	Analyse en Composantes Principales
AP	Average Precision
CAO	Conception Assistée par Ordinateur
CE	Cross-Entropy
CGAN	Conditional Generative Adversarial Networks
CT-GAN	Consistency regularized GAN
CNN	Convolutional Neural Network
Chap.	Chapitre
CUDA	Compute Unified Device Architecture
CVPR	Computer Vision and Pattern Recognition
DCGAN	Deep Convolutional Generative Adversarial Networks
DET	Detection Error Tradeoff
DPM	Deformable Parts Model
D2D	Distribution to Distribution
ECCV	European Conference of Computer Vision
EM	Expectation Maximisation
FID	Frechet Inception Distance
Fig.	Figure
FN	Faux Négatif
FP	Faux Positif
FPPF	Faux Positif Par Fenêtres
FPPI	Faux Positif Par Image
GAN	Generative Adversarial Networks
GIL	Global Interpreter Lock
GM	Gaussian Mixture
GPU	Graphical Processing Unit
GRA	GRAdient normalisé
HOG	Histogram of Oriented Gradients / Histogramme de gradients orientés
ICCV	International Conference of Computer Vision
ISPRS	International Society for Photogrammetry and Remote Sensing

KID	Kernel Inception Distance
LAD	Least Absolute Deviation
LBP	Local Binary Pattern
LSE	Least Squares Error
LTP	Local Ternary Pattern
mAP	Average Precision moyenne
MLP	Multi-Layers Perceptron / Perceptron Multi-couches
MR	Miss Rate / Taux d'erreur
MRF	Markov Random Field
MSE	Mean Square Error
NDG	Niveaux De Gris normalisés
NMS	Non-Maximum Suppression / Suppression des Non-Maxima
OIRDS	Overhead Imagery Research DataSet
OIRDS-EGI	OIRDS-Ensemble de Grandes Images
OIRDS-EPI	OIRDS-Ensemble de Petites Images
OP	Operating Point / Point Opérationnel
P	Nombre de Positifs
Prec	Précision
P2P	Point to Point
RAM	Random Access Memory
RBF	Radial Basis Function
Rec	Recall / Rappel
R-CNN	Region Convolutional Neural Network
RER-CNN	Rotation Equivariant Region CNN
ROC	Receiver Operator Characteristic
RoI	Region of Interest
SAGAN	Self-Attention GAN
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Feature
SVM	Support Vector Machine / Séparateur à Vaste Marge
Tab.	Table
TM	Template Matching
TVP	Taux de Vrais Positifs
TFP	Taux de Faux Positifs
VAE	Variational Auto Encoder
VeDAI	Vehicule Detection in Aerial Imagery
VeDAI-GIC	VeDAI, Grandes Images Couleurs
VeDAI-GII	VeDAI, Grandes Images Infrarouges
VeDAI-PIC	VeDAI, Petites Images Couleurs
VeDAI-PII	VeDAI, Petites Images Infrarouges
VN	Vrai Négatif
VP	Vrai Positif
WGAN	Wasserstein GAN
WGAN-GP	Wassertein GAN with Gradient Penalty
WGSS	Within Group Scatter Sum, distance intra-cluster
XML	Extensible Markup Language

Anglicisme	Explication
fine-tuner/fine-tuning	ajuster les poids d'un réseau CNN à partir d'une initialisation
pool/pooling/pooler	mise en commun/calcul d'une statistique locale sur un voisinage
warper	distordre une image en lui appliquant une transformation affine
dropout	non-utilisation d'un neurone
fold	découpe d'un ensemble d'images entre apprentissage, validation et test
baseline	approche rudimentaire que l'on souhaite dépasser
pipeline	système complet de détection
hard-mining	processus de sélection d'exemples
bootstrapping	processus itératif de collecte des exemples
back-propagation/backward	Calcul du gradient d'une structure CNN
forward	Calcul de la sortie d'un CNN
blending	fondu d'une image dans une autre
inpainting	recréer une zone définie d'une image
thread	fil d'exécution ou tâche
warp	ensemble de 32 threads utilisé dans les GPUs

Abstract

Abstract

The following manuscript is an attempt to tackle the problem of small vehicles detection in vertical aerial imagery through the use of deep learning algorithms. The specificities of the matter allows the use of innovative techniques leveraging the invariance and self similarities of automobiles/planes vehicles seen from the sky. We will start by a thorough study of single shot detectors. Building on that we will examine the effect of adding multiple stages to the detection decision process. Finally we will try to come to grips with the domain adaptation problem in detection through the generation of better looking synthetic data and its use in the training process of these detectors.

Résumé

Cette thèse présente une tentative d'approche du problème de la détection et discrimination des petits véhicules dans des images aériennes en vue verticale par l'utilisation de techniques issues de l'apprentissage profond ou "deep-learning". Le caractère spécifique du problème permet d'utiliser des techniques originales mettant à profit les invariances des automobiles et autres avions vus du ciel. Nous commencerons par une étude systématique des détecteurs dits "single-shot", pour ensuite analyser l'apport des systèmes à plusieurs étages de décision sur les performances de détection. Enfin nous essayerons de résoudre le problème de l'adaptation de domaine à travers la génération de données synthétiques toujours plus réalistes, et son utilisation dans l'apprentissage de ces détecteurs.

Remerciements

Je tiens avant tout à remercier les membres de mon jury, professeurs titulaires, Elisa Fromont et Sébastien Lefèvre pour avoir rédigé des rapports détaillés et d'une grande qualité permettant d'améliorer grandement ce manuscrit, Matthieu Cord, professeur des Universités et président du jury, pour avoir accepté d'héberger ma soutenance et pour avoir, entre autres, pris ma défense, ainsi que mes encadrants Frédéric Jurie, professeur des Universités et Farid Oudyi, expert Safran ED, pour leur soutien. Les questions très pertinentes du jury ont permis, en me poussant dans mes retranchements, de faire émerger des discussions profondes et intéressantes mettant en relief le travail accompli.

Une énorme partie de ces remerciements doivent aller à Frédéric Jurie, qui, malgré ses nombreuses obligations, a toujours su trouver le temps pour m'écouter et m'orienter grace à sa fabuleuse maîtrise du domaine et ses immenses qualités humaines.

Merci à mes collègues, notamment au docteur Constant Guillot, qui m'a supporté longuement et apporté son soutien précieux en C++.

Merci à mon ami Hedi Ben Younes pour son optimisme et sa gentillesse, qui m'ont suivi pendant bien plus que ces 3 ans de thèse.

Merci au docteur Taylor Mordan pour m'avoir, non seulement aidé dans mes recherches en ce qui concerne le choix de l'interpolation bilinéaire, qui a conduit au RoI-Align tournant, mais aussi pour ses modèles Tikz (, qui m'ont aidé à construire la figure 4.9) et surtout pour son amitié.

Merci au docteur Alexis Lechervy, qui est une des têtes pensantes derrière une partie des idées et des expériences présentes dans le Chapitre 3.

Merci aux administrateurs réseaux du GREYC pour avoir toujours répondu à mes questions dans les plus brefs délais.

Merci à mes parents pour m'avoir aidé à corriger une grande partie des fautes d'orthographe de ce manuscrit.

Enfin je voudrais conclure ces remerciements formels en m'adressant à des personnes, qui n'ont pas directement de lien avec mes travaux de thèse mais que j'aimerais cependant mentionner.

Merci à Fabien, Luc, Augustin et Aurélien les prestataires, qui m'ont accompagné par leur humour et leurs attentions durant ces 3 ans.

Merci à Caroline Cochet-Escartin et au docteur Jérôme Lambert pour m'avoir décidé à m'engager dans cette thèse.

Merci à tous mes autres amis en particulier ceux du groupe Supérette central.

Merci enfin à celle pour qui mon coeur bat tous les jours !

Chapitre 1

Introduction à la détection d'objets dans des images aériennes

À travers le bruit médiatique permanent sur les progrès de l'Intelligence Artificielle et les réseaux neuronaux, il nous paraît souhaitable de prendre le temps d'étudier quelles sont les réelles capacités de ces systèmes dits "intelligents" sur des problèmes simples – en tout cas pour nous humains – comme *la détection automatique d'objets sur les images*.

Toute tentative de résolution de cette tâche, appartenant au domaine de la vision par ordinateur, implique en premier lieu l'aptitude du système à reconnaître des formes et des couleurs dans une image. Le système doit aussi identifier les associations de ces différentes formes et couleurs avec les caractéristiques visuelles des objets (par exemple, si des yeux, une bouche et un nez, alors c'est un visage). En cela, le deep-learning a permis des avancées considérables sur les alternatives existantes. Nous verrons pourquoi et comment.

La seconde partie de tels systèmes, bien plus discutée dans les médias compte tenu de ses implications éthiques et philosophiques, vise à permettre de prendre une décision finale importante et à conséquences réelles (comme diagnostiquer un patient, donner un coup de volant ou orienter un missile) à partir de ces informations. Cela nous semble

cependant bien plus complexe et hors de la portée du seul deep-learning ou en tout cas certainement de cette thèse.

Nous nous limitons dans cette thèse au problème passionnant de la construction de systèmes de perception adaptés à la détection des petits véhicules vus du ciel de manière verticale (comme sur les images GoogleEarth).

Cette thèse a été réalisée au sein du laboratoire GREYC [1] dans l'équipe Image ainsi que dans la société Safran Electronics and Defense, dans le cadre d'un contrat CIFRE de 3 ans. Le groupe Safran est spécialisé aussi bien dans l'aéronautique et le spatial que dans les technologies liées à la défense. L'ensemble du Groupe Safran, étant impacté par ces nouvelles technologies, a décidé de monter en compétences sur le sujet par le biais de plusieurs doctorants CIFRE et de la création de centres d'excellence comme SafranTech.

Ce premier chapitre introduit les concepts et problématiques de la détection d'objets nécessaires à la bonne compréhension de la thèse, ainsi que les particularités de la détection de petits véhicules en imagerie aérienne.

1.1 Objets visuels et détection

1.1.1 Objets visuels et boîtes englobantes

Nous commençons cette thèse par l'aveu de notre impossibilité à définir une notion universelle d'"objets visuels". Cette impossibilité a plusieurs causes profondes. La première vient de la notion même d'objet : le ciel est-il un objet de la même manière qu'un piéton (les Anglais possèdent dans leur vocabulaire une notion non présente en français de "stuff" et de "thing" pour exprimer cette distinction) ? La deuxième, plus terre à terre, vient de la différence entre la représentation de l'image en termes de grille de pixels et le concept abstrait d'entité visuelle ou objet. Smeulders et al. [219] introduit le terme "fossé sémantique" pour parler de la différence entre l'information effectivement présente dans une image et l'interprétation qui peut en être faite par différents utilisateurs dans différentes conditions.

La notion d'objets est donc définie concrètement, dans le cas supervisé, au cas par cas,

1. Laboratoire de l'Université de Caen Normandie et de l'ENSICAEN, UMR 6602 du CNRS

pour chaque cas applicatif, au moyen d'exemples annotés constituant une *base d'images* ou *base de données image*. Par exemple, pour une tâche donnée, le concept de "bateau" peut être uniquement relatif à la coque et non au mât ou aux voiles ; pour une autre cela pourra être différent.

Ce travail de création de bases de données, une fois le choix des images fait, passe par une phase de "labellisation" ou "d'annotation", qui revient dans sa forme la plus complexe à assigner à chaque pixel de l'image un type d'objet (par exemple : ce pixel est un bateau, celui-ci non).

A ces difficultés d'ordre conceptuel, vient s'ajouter un bruit sur les annotations. Une même personne, utilisant donc une définition de concept a priori cohérente, peut labelliser différemment les pixels d'une même image qui lui serait présentée plusieurs fois.

En général, en détection d'objets, pour pallier au bruit d'annotation et parce que cela est plus simple à construire, il est choisit de recourir à une approximation grossière appelée boîte englobante ($\mathcal{B}_{box}$) et définie comme étant le plus petit rectangle aligné avec les axes de l'image contenant tous les pixels d'un objet. Cette définition s'est popularisée à travers les compétitions de détection d'objets Pascal-VOC [55], Imagenet [44], MS-Coco [143] et maintenant OpenImages [133], pour ne citer que quelques exemples. Elle est aussi bruitée, pour les mêmes raisons, mais beaucoup moins du fait du nombre restreint de degrés de liberté, et a l'avantage d'être générale et d'admettre une paramétrisation simple :

$$\mathcal{B}_{box} = (x_{min}, x_{max}, y_{min}, y_{max}) \tag{1.1}$$

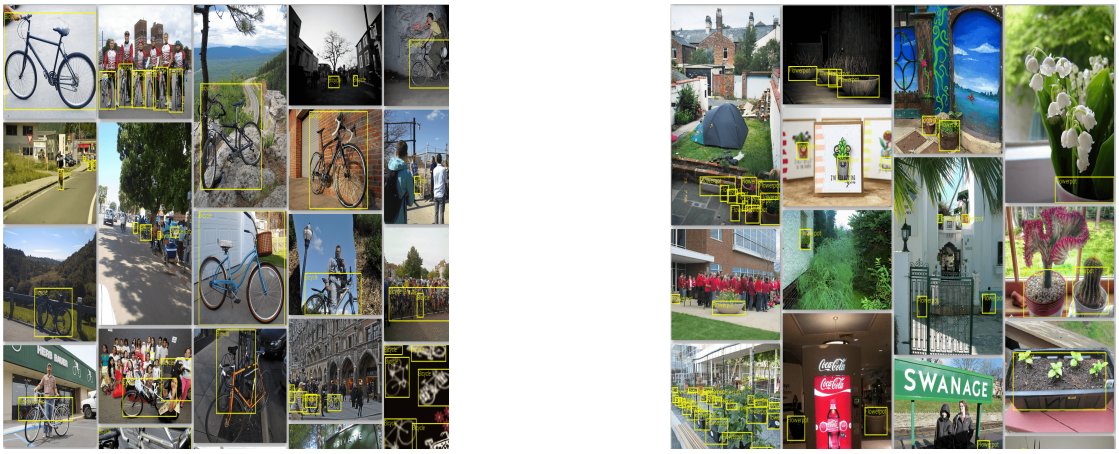
exprimée en coordonnées image. Ainsi il est possible de définir $W = x_{max} - x_{min}$ la longueur de la boîte, $H = y_{max} - y_{min}$ sa largeur, et son centre de coordonnées ($x_{center} = \frac{x_{max}+x_{min}}{2}, y_{center} = \frac{y_{max}+y_{min}}{2}$).

Si les différents objets contenus dans les images sont uniques, mais sont présents dans des positions et éclairages différents (variabilité intra-objet), nous parlons alors de **détection d'instances**.

Si, au contraire, ils appartiennent à une catégorie générale d'objets (ou classe) comme

"bicyclette" ou "plante en pots" (voir Figure 1.1), qui présente une encore plus grande *variabilité intra-classe* ; nous parlons alors vraiment de **détection d'objets**.

C'est ce cadre de détection de classes d'objets qui nous intéressera tout au long de cette thèse.



(a) Bicyclettes (b) Plantes en pots

FIGURE 1.1 – Exemples de vérités terrains annotées prises dans OpenImages [133].

Lorsque les images sont annotées avec des boîtes, il est attendu que les détecteurs produisent des détections sous forme d'ensembles de boîtes verticales (c'est un abus de langage pour dire aligné avec les axes) par classe, avec en général, pour chaque boîte, le degré de confiance du détecteur dans sa réponse.

Nous disposons alors, pour évaluer leurs performances en détection, de listes ordonnées par confiance décroissante de coordonnées de boîtes par classe. Il est alors possible d'imaginer des métriques permettant de mesurer la proportion des objets effectivement présents dans la scène (ce sera d'ailleurs fait dans la section suivante). Cela suppose bien sûr de disposer au préalable d'images annotées : comme nous venons de le voir plus haut, il s'agit d'images associées à des listes de boîtes de classes indiquées comme étant "vraies" ; nous parlons alors de boîtes "vérités terrain".

Ce problème évident des données disponibles pose beaucoup plus de questions qu'il n'y paraît, et sera abordé plus en détail dans le Chapitre 4 ; nous faisons ici l'hypothèse que la vérité terrain liée aux images est disponible.

1.1.2 Produire des boîtes englobantes à l'aide de l'apprentissage machine

Pour être capables d'inférer la position des objets dans les images, les détecteurs actuels utilisent des techniques de l'apprentissage statistique (ou apprentissage-machine). Nous faisons l'hypothèse que le lecteur est familier avec les bases de l'apprentissage statistique [2] et nous nous contentons ici de rappeler quelques principes clés.

Le but de l'apprentissage supervisé classique est de trouver dans un espace de fonctions donné (paramétrique ou non) une fonction f interpolante sur un ensemble de points (x, y) issus d'un processus aléatoire de loi inconnue $P(X, Y)$ appelé : l'ensemble d'apprentissage, de manière à ce qu'elle puisse généraliser sur d'autres données non vues $(x, ?)$ issues du même processus (ensemble de test).

Dans notre cas, les x sont les objets visuels définis dans la section précédente c'est-à-dire des vignettes (c'est-à-dire des morceaux d'une image entière) contenant ou non un objet et les y, des étiquettes de classes ou labels : nous disons que nous entraînons notre modèle en classification (par opposition à la régression de grandeurs scalaires). Nous parlons d'apprentissage (ou d'entraînement) lorsque nous ajustons notre modèle sur des points connus (avancement représenté par un coût d'attache aux données) et d'inférence lorsqu'il s'agit de faire une prédiction sur un nouvel échantillon.

Lorsque nous pouvons formuler l'attache aux données par un coût différentiable et que f est aussi différentiable (sous-différentiable en fait) par rapport à ses paramètres, nous pouvons utiliser pour l'apprentissage tout un arsenal de méthodes d'optimisation utilisant le gradient ainsi parfois que des dérivées d'ordre supérieur afin d'ajuster petit à petit les sorties de la fonction pour qu'elle passe ainsi au plus près des exemples d'apprentissage.

Le deep learning en classification et détection d'objets (et autres réseaux de neurones) s'inscrit quasi complètement dans cette veine, faisant même dire à Judea Pearl, un expert du domaine, que toutes les dernières avancées en ce domaine ne sont rien d'autre que du "Curve fitting"[85].

La figure 1.2 montre les prédictions faites par différents modèles avant et après en-

2. un bon ouvrage de référence sur le sujet est le livre de Bishop [13]

traînement en régression par descente de gradient. Le processus étudié est : $P(Y|X = x) \sim \mathcal{N}(\sin(x), \epsilon^2)$ et le coût d'attache aux données : la moyenne du carré des erreurs (moindres carrés).

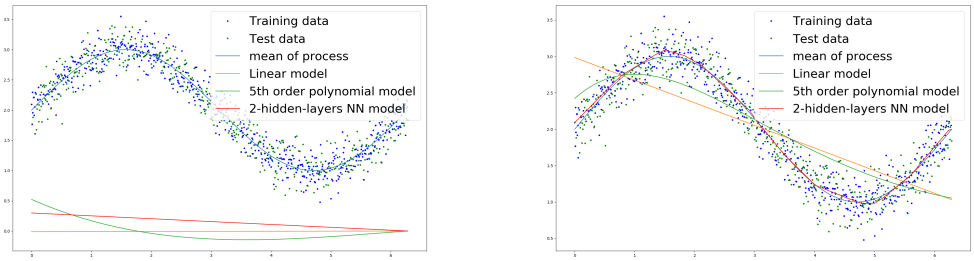


FIGURE 1.2 – Ajustement de modèles de complexité différente sur un ensemble d'apprentissage : à gauche les modèles avant l'apprentissage à droite après

La détection d'objets peut être vue comme l'application d'un classifieur sur toutes les vignettes [3] des images. Le passage des points de l'exemple précédent à la classification de vignettes revient juste à un effort d'abstraction supplémentaire :

— les entrées de f ne sont plus scalaires, mais au contraire des vecteurs de taille fixe (de taille 1200 pour une vignette couleur de 20x20 pixels par exemple).

— Les sorties de f représentent un score abstrait de classe pouvant être aussi bien scalaire que multidimensionnel en fonction de la formulation du classifieur et de la multiplicité des classes étudiées.

Nous choisissons de ne pas faire ici de liste exhaustive des différents choix possibles de f et du coût ainsi que des différentes méthodes d'optimisation.

Enfin, pour passer de la classification de vignette à la production d'une liste ordonnée de boîtes de classification sur l'image, le plus simple est d'effectuer un "parcours par fenêtres glissantes".

Nous entraînons d'abord notre classifieur sur des vignettes de taille fixe (convertibles en vecteurs 1D en aplatissant), classées comme objets et non objets (fonds). Dans un second temps, l'image est parcourue exhaustivement de haut en bas et de gauche à droite, de manière systématique, en classifiant toutes les vignettes successivement, et

3. zones rectangulaires contenues dans l'image

ce, à différentes échelles pour avoir différentes tailles de fenêtres. Ce procédé s'appelle "parcours par fenêtres glissantes sur pyramide d'images", car les différentes résolutions d'images utilisées forment une pyramide (les images sont de plus en plus petites à mesure que l'on zoome sur l'image).

1.1.3 Évaluation des détecteurs d'objets

Nous disposons maintenant des boîtes vérités terrains des images et d'un moyen de produire une liste de boîtes de classification avec leur score. Pour évaluer un détecteur, il faut être capable de vérifier si chaque boîte trouvée par le détecteur "correspond" bien à une "vraie" boîte (annotations). Ce travail fait pour chaque boîte, la performance du détecteur peut ensuite être calculée. Ces deux points sont traités dans les deux sous-sections qui suivent.

Vérifier la validité d'une boîte inférée

Considérons ici le cas d'une image possédant un objet et d'une liste de boîtes prédites par un détecteur. En fonction des impératifs des différents utilisateurs du système de détection, il existe différents moyens de dire si oui ou non un objet est bien présent dans chaque boîte inférée, en utilisant la vérité terrain.

Un premier critère naïf est de raisonner en termes de position uniquement ; si le centre d'une fenêtre de détection (indice d) est dans un disque d'un rayon R fixé depuis le vrai centre de l'objet (indice t) indiqué par la boîte de vérité terrain, alors la boîte est considérée comme contenant un objet.

$$x^d_{center}, y^d_{center} \in D(x^t_{center}, y^t_{center}, R) \tag{1.2}$$

Ce critère, dans le cas où la taille des boîtes produites est importante pour l'utilisateur, possède des limitations : une boîte trouvée par le détecteur peut alors être beaucoup plus grande ou au contraire beaucoup plus petite que l'objet, et pourtant satisfaire pleinement ce critère basé sur le centre, comme le montre la Figure 1.3.

Une alternative est de vérifier en plus de la position du centre, la différence des

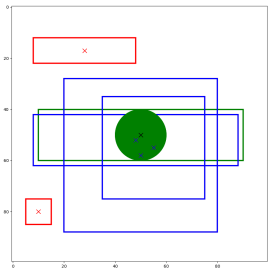


FIGURE 1.3 – La boîte-vérité terrain est en vert, les détections considérées comme fausses selon **le premier critère** sont en rouge, celles valides en bleu

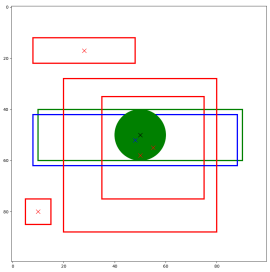


FIGURE 1.4 – Deux boîtes jusqu'alors comptées comme des détections valides, se sont faites rejetées par le critère sur les longueurs. Il ne reste plus qu'une seule bonne détection.

longueurs et largeurs entre la boîte détectée et la vraie boîte. Par exemple, une boîte peut être considérée comme une bonne détection si :

$$x^d_{center}, y^d_{center} \in D(x^t_{center}, y^t_{center}, R), |W^d - W^t| < s_W, |H^d - H^t| < s_H \tag{1.3}$$

Nous utilisons les notations de la section précédente. R, s_W et s_H sont des paramètres utilisateurs fixés, dans l'exemple de la Figure 1.4, tous à la valeur de 10.

Malheureusement, si les boîtes à détecter sont très grandes, une différence relative minime en termes de longueur ou de largeur pourrait signifier une différence absolue très grande et donc une bonne détection pourrait être considérée comme erronée. Cela introduirait une différence de traitement entre les boîtes de différentes tailles, ce qui n'est pas (forcément) souhaitable. Pour pallier à cela, nous pourrions normaliser par la taille de la vérité terrain pour avoir une mesure invariante :

$$x^d_{center}, y^d_{center} \in D(x^t_{center}, y^t_{center}, R), \frac{|W^d - W^t|}{W^t} < s'_W, \frac{|H^d - H^t|}{H^t} < s'_H \tag{1.4}$$

Nous pourrions inventer d'autres critères plus subtils ou plus appropriés pour telle ou telle application, mais celui qui fait presque l'unanimité dans la communauté de vision par ordinateur est le critère de Jaccard ou littéralement critère d'IoU (Intersection over Union) entre nos deux boîtes $\mathcal{B}^d$ et $\mathcal{B}^t$. Ce critère est défini, **dans le cas d'une**

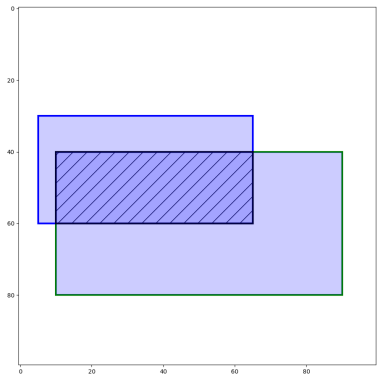


FIGURE 1.5 – IoU entre deux boîtes : l'intersection est hachurée, et l'union est en bleu transparent

intersection non-nulle des boîtes par :

$$J(\mathcal{B}^d, \mathcal{B}^t) = \frac{\mathcal{A}_{\mathcal{B}^d \cap \mathcal{B}^t}}{\mathcal{A}_{\mathcal{B}^d \cup \mathcal{B}^t}} \tag{1.5}$$

où $\mathcal{A}$ désigne l'aire. Il est présenté Figure 1.5.

Il se traduit analytiquement dans le cas de nos boîtes verticales par :

$$J(\mathcal{B}^d, \mathcal{B}^t) = \frac{\mathcal{A}_{intersection}}{W^d * H^d + W^t * H^t - \mathcal{A}_{intersection}} \tag{1.6}$$

Avec l'aire de l'intersection :

$$\mathcal{A}_{intersection} = \Big(min(x^d_{max}, x^t_{max}) - max(x^d_{min}, x^t_{min})\Big) * \Big(min(y^d_{max}, y^t_{max}) - max(y^d_{min}, y^t_{min})\Big) \tag{1.7}$$

Cette formulation a, en plus, l'avantage d'être intuitive ainsi que d'être trivialement "vectorisable" pour plusieurs boîtes de détection et plusieurs vérités terrain.

Il est alors possible de définir plusieurs seuils de détection de l'objet selon les besoins : si les besoins sont d'avoir une localisation très précise, nous imposons un IoU supérieur à $0.7 - 0.8$ ou si au contraire nous voulons une localisation plus flexible, un IoU de $0.5 - 0.3$ suffira. Une bonne performance à un IoU grand implique forcément une performance

encore meilleure sur les autres IoU moins élevés.

Évaluer les performances des détecteurs

Après avoir défini un critère permettant de dire si chaque boîte est une détection correcte ou non, il est dorénavant possible d'étendre ce résultat pour un nombre arbitraire d'objets par image, de boîtes, et d'images. Un bon détecteur doit assurer deux objectifs complémentaires. Il doit, pour une classe donnée, trouver *tous les objets de cette classe présents dans la base*. Nous dirons que le rappel R doit être élevé. Mais il doit aussi *ne pas faire trop d'erreurs* en trouvant des objets là où il n'y en a pas ou en trouvant plusieurs fois le même objet : nous souhaitons que la précision P soit également élevée. Plus formellement :

$$R = \frac{\#bonnes\ détections}{\#objets\ présents}$$
$$P = \frac{\#bonnes\ détections}{\#détections}$$

Il est bien entendu facile de construire un détecteur de rappel à 1 en sortant une infinité de fenêtres, ou de précision à 1 en ne détectant que les objets très faciles dont le détecteur est sûr. Mais il est très difficile de construire un détecteur alliant bonne précision et bon rappel.

Si les boîtes de sorties de notre détecteur sont ordonnées, nous pouvons définir autant de détecteurs qu'il y a de valeurs de score présentes. Par exemple, si les scores sont normalisés et que nous ne prenons en compte que les boîtes les plus confiantes à score élevé, nous allons avoir un détecteur plus précis, mais nous risquons de rater des cibles ; à l'inverse, en mettant un seuil faible tous les véhicules seront probablement trouvés, mais la précision risque de chuter avec l'augmentation du nombre de faux positifs.

De ce fait, pour noter un détecteur, il est commun d'effectuer une moyenne de la précision obtenue pour différents sous-détecteurs obtenus pour des seuils différents et donnant donc des rappels différents : c'est ce qui est appelé l'AP (Average Precision). Elle correspond aussi à une approximation de l'aire sous la *courbe Précision-Rappel* présentant les différents sous-détecteurs pouvant être obtenus dans le plan Précision

(ordonnée)-Rappel (abscisse). C'est donc une courbe de tendance décroissante : à mesure que le seuil du détecteur est baissé, le rappel augmente (déplacement vers la droite de la courbe) et la Précision peut chuter si des faux-positifs sont découverts ou augmenter si d'autres vrais objets apparaissent. Cela explique la forme en dents de scie, caractéristique des courbes Précision-Rappel, que nous observerons dans la suite du manuscrit.

Selon la méthodologie introduite par PASCAL-VOC [54], que nous utiliserons dans toute la thèse, l'AP se mesure sur 11 points également répartis entre un rappel de 0 et un rappel de 1 : {0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0}.

En pratique ces points sont interpolés, car il n'existe en général pas de seuil donnant exactement leur valeur.

Liés à la précision, nous définissons aussi le nombre de Faux Positifs (détections incorrectes d'un objet) Par Image ou FPPI.

Nous parlerons aussi de mAP (mean Average Precision) quand l'AP est moyennée sur des ensembles ou sur des classes.

Une problématique importante est de savoir dans quelle mesure, les performances obtenues sur la base de données (BDD) considérée sont représentatives de la performance réelle.

Ainsi pour justifier de la robustesse statistique de la méthode, il est commun de réaliser des validations-croisées. C'est-à-dire que l'AP est calculée en entraînant le détecteur sur une partie des données puis en testant sur l'autre, et en répétant le procédé suivant différentes découpes (folds), afin de décorréler les performances de la découpe considérée. Ce procédé est recommandé lorsque les BDDs utilisées sont de petites tailles.

Nous préférons les métriques basées sur l'AP aux critères basés sur l'aire sous la courbe ROC (AUC) [13] car, en cas de fort déséquilibre des classes, le taux de faux positifs n'est pas informatif si le nombre de vrais négatifs est trop grand.

La littérature du domaine propose de nombreux autres critères d'évaluation de détecteurs, comme la rapidité d'exécution par exemple ou l'occupation mémoire du système. Mais dans cette thèse, l'accent sera donné sur la recherche de performances de détection en termes d'AP.

1.2 Détection d'objets en imagerie aérienne : métriques et particularités

Dans le cas de l'imagerie aérienne, les concepts évoqués dans la section précédente doivent être précisés.

1.2.1 Des différences profondes avec l'imagerie traditionnelle

L'imagerie aérienne propose souvent des images peu résolues sur des bandes spectrales parfois différentes du spectre visible (IR bande 1 ou 2). Ces images peuvent être bruitées par des nuages et autres intempéries, qui affectent les capacités du capteur. Les reflets du soleil sur les surfaces métalliques ou réfléchissantes font parfois saturer le capteur et apparaître des tâches artificielles. Les ombres portées cachent les objets comme les voitures en diminuant leur contraste. Pire, la végétation ou la signalétique occultent parfois une grande portion du véhicule à détecter. De plus, l'obliquité, c'est-à-dire l'angle réel entre le capteur et la scène n'est souvent pas idéale (90 degrés entre le sol et la ligne de visée), ce qui oblige à traiter les images par un mécanisme, qui est appelé *l'ortho-rectification*. Cet algorithme permet, à l'aide de modèles de reliefs de la carte et des caractéristiques du capteur, de corriger l'image pour qu'elle apparaisse comme si elle était vue depuis la verticale. Bien sûr ce processus n'étant pas parfait, cela introduit des artefacts (allongements, mosaïquage). Un exemple est montré Figure 1.6. Il faut remarquer aussi que, le but étant souvent de capturer une large surface de terrain, les véhicules n'occupent qu'une petite portion de l'image sauf dans les zones urbaines particulièrement denses (parkings, rues, etc.), mais même dans ce cas chaque voiture n'occupe que très peu des pixels de l'image. Ces conditions particulières de prise de vue induisent d'énormes écarts de statistiques sur la distribution des boîtes-vérité terrain, en comparaison avec les bases de données de détection d'objets de la vie de tous les jours. Ces différences sont résumées Table 1.1.

Afin de construire un bon détecteur de véhicules en imagerie aérienne, nous utiliserons, dans une optique bayésienne, tout l'a priori possible (quitte à perdre en généralité). Les difficultés ne sont donc pas les mêmes qu'avec les bases de données classiques, nous

FIGURE 1.6 – Image ortho-rectifiée issue de Wikipedia

Bases d'Imagerie aérienne	Bases traditionnelles (VOC, COCO, etc.)
petits objets	objets de toutes tailles
beaucoup de fond	peu de fond
objets de toutes orientations	objets tous plus ou moins verticaux
peu de variabilité intra-classe à orientation fixée	beaucoup de variabilité intra-classe
peu de variabilité inter-classe	grande variabilité inter-classe
faible résolution	bonne résolution

TABLE 1.1 – Différences entre les bases de données usuelles en traitement d'images et notre cas d'application, les inconvénients sont marqués en rouge et les avantages en vert

y reviendrons. Par exemple, nous aurons moins à gérer de changements de gabarits de boîtes. Nous aurons en revanche des classes très similaires les unes des autres (p. ex. pickups et voitures). Il sera donc difficile de les différencier les unes des autres. La dimension faible des voitures jouera aussi contre nous, de même que le déséquilibre des classes. Ce tableau guidera nos choix de design tout au long de la thèse et pourra servir d'inspiration au développement de nouvelles méthodes dans de futurs travaux.

En vertu de ces observations et compte tenu des domaines applicatifs visés par cette thèse, après avoir présenté des exemples de bases, nous définissons un autre critère de détection dans la lignée de ceux présentés dans la section précédente.

1.2.2 Des BDDs diverses et variées

Très récemment, il y a eu un regain d'intérêt de la communauté scientifique pour la détection de petits véhicules en imagerie aérienne. Nous faisons ici une liste quasi exhaustive des bases de données existantes. Ces bases sont utiles pour entraîner des algorithmes de détection ou pour comparer leurs performances en imagerie aérienne.

GoogleEarth[94] : Elle présente des images de la ville de Bruxelles. Malheureusement son intérêt est limité tant le nombre d'images est faible. D'autre part, le manque de variabilité des véhicules présents provoque des risques de sur-apprentissage. Il y a en tout 30 images, dont 15 avec des annotations angulaires [95], une seule classe véhicule. La mAP est prise avec un critère Pascal VOC sur l'IoU sur 5 folds. L'état de l'art est détenu par RIFD-CNN[28] avec 94.6% mAP.

Munich (DRL3K)[141] : Cette base présente plusieurs intérêts, de par :

- son caractère multi-classes
- son grand nombre de véhicules
- sa grande variabilité de fonds

Il y a 20 grandes images de taille 5616∗3744 prises à 1000m du sol avec plusieurs classes, 10 images d'apprentissage avec 3500 voitures et 70 camions et 10 images de test avec 5800 voitures et 90 camions. D'autres classes sont aussi disponibles comme des remorques ainsi que de la signalisation au sol. En général les auteurs choisissent leur propre métrique même si la plus utilisée est le score F1 maximal, moyenne géométrique du rappel et de la précision obtenue pour le meilleur seuil de détection, moyenné sur les deux classes. [230] domine la littérature avec 83% de score F1. Du même auteur [229] atteint 82%, mais donne des boîtes orientées. D'autres articles dignes d'être mentionnés sont [220, 221, 45].

OIRDS [231] : Avec seulement 180 véhicules, cette base n'est quasiment pas utilisée dans la communauté.

VeDAI [189] : La contribution de VeDAI est d'introduire des images infrarouges, faiblement résolues et surtout des véhicules présents dans des environnements non ur-

bains et plus variés (déserts, forêt, etc.), ce qui coïncide avec nos besoins. Les images sont extraites de la base OIRDS et des collections d'images de l'Utah. Il y a 11 classes de véhicules. VEDAI sera notre principal terrain de jeu, en tout cas dans les premiers chapitres.

COWC [168] : Introduit récemment à ECCV2016 c'est une très grande base de données, contenant jusqu'à 32000 voitures. L'auteur fournit aussi 60000 patches de "négatifs difficiles" choisis à la main, ce qui est intéressant lorsque qu'un détecteur est entrainé sans stratégie particulière de bootsrapping. Malheureusement il n'y a pas d'annotations de l'ensemble de test ce qui rend les comparaisons de détecteurs difficiles.

DOTA [247] : Sortie en 2018 à CVPR c'est la première base de données (BDD) d'imagerie aérienne à incorporer une métrique des boîtes orientées en plus de la métrique de boîtes verticales. Il y a 2800 images avec près de 200000 instances et 15 catégories. Cette BDD deviendra sûrement une référence avec les années au même titre que VeDAI le fut en son temps. Le classement en ligne montre que les structures de type mask R-CNN sont les meilleures pour cette tâche pour le moment avec un état de l'art culminant à 76.2% de mAP mais il n' y a pas encore de publications à l'heure d'écriture de ces lignes. UCAS-AOD[266], NWPU VHR10[29] et HRSC2016[153] donnent aussi des annotations orientées, mais leur usage reste marginal.

xView [136] Cette énorme BDD avec 60 classes et plus d' un million d'instances a été récemment ouverte dans le cadre d'une compétition par le Pentagone.

En plus des bases que nous venons de voir, les bases "Vaihingen" et "Toronto" [117], composées d'images de l'ISPRS, semblent aussi être le théâtre d'une compétition de détection de véhicules, mais sont surtout connues pour la compétition de segmentation sémantique associée. De plus, la DGA organise en ce moment des compétitions semi-ouvertes de détection de petits véhicules [1] pour faciliter le travail des Interpréteurs Images (IP).

À la lumière de toutes ces bases, les nouveaux venus dans le milieu de la détection en imagerie aérienne peuvent être frappés par le manque de standard en termes de

définition d'une bonne détection aussi bien qu'en termes d'évaluation des détecteurs. Il serait profitable que la communauté s'organise et s'accorde sur la marche à adopter pour éviter les travaux utilisant des métriques et des protocoles d'évaluation tous différents sur les mêmes bases de données. Les compétitions dans lesquelles les soumissions de résultats sont validées informatiquement par un serveur commun sont un bon moyen d'aboutir à des standards c'est pourquoi DOTA, Vaihingen et xView sont de bonnes initiatives, mais elles n'ont pas (encore) l'attrait des compétitions de type Pascal-VOC [55].

Des exemples des différentes images de la plupart de ces bases sont disséminés dans le présent mémoire de thèse.

1.2.3 La métrique VeDAI et son utilité dans l'imagerie aérienne

Lors de travaux précédents [189], Razakarivony et Jurie, ont développé une métrique répondant aux impératifs de nos applications et plus largement aux problématiques posées en imagerie aérienne. Partant du principe, sur lequel nous reviendrons très largement au Chapitre 4, que les boîtes verticales sont des approximations très pauvres dans le cadre de véhicules pouvant tourner librement. Ils définissent une bonne détection en termes de position du centre de la boîte de détection par rapport à l'ellipse inscrite dans la boîte orientée "vérité terrain" du véhicule visé : si le centre de la détection appartient à cette ellipse, la détection est comptée comme correcte, sinon non (voir la Figure 1.7). Il n'y a donc ici pas de notion de taille. Son intérêt tient aussi au fait que dans cette nouvelle base relative à l'ellipse courante, il est aisé de calculer différentes lignes de niveau associées à différentes tolérances en termes de précision de localisation. Ceci est important en détection car le caractère discret de la grille sur laquelle est appliqué le classifieur peut nuire au rappel.

1.3 Organisation du document

Maintenant que les concepts de base sont posés, nous pouvons plonger dans le coeur du sujet en construisant et évaluant les performances de détecteurs d'objets dans le cadre particulier dont les spécificités viennent d'être brossées. Le Chapitre 2 servira, en

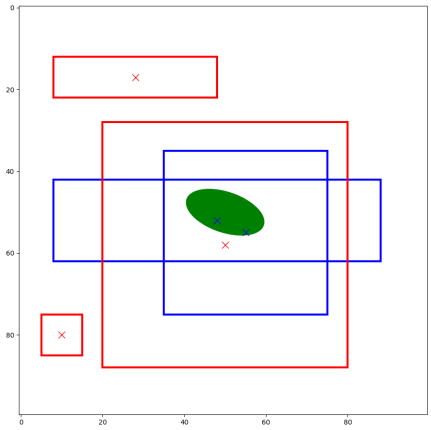


FIGURE 1.7 – La boîte-vérité terrain est en vert, les détections considérées comme fausses selon **le premier critère** sont en rouges, celles valides en bleues

dépassant le paradigme de la fenêtre glissante, à construire un détecteur "single-shot" simple, dont l'étude permettra de dégager quelques règles générales sur l'édification de ces systèmes. Il sera aussi l'occasion de comparer ce détecteur simple à d'autres implémentations "état de l'art" de la littérature. Le troisième chapitre introduira une première cascade simple présentant une approche de la régression originale. Il étudiera de plus près l'estimation d'orientation et son apport pour la détection d'objets. Le chapitre, qui suivra, permettra, grâce à l'introduction de boites orientées dans un algorithme "état de l'art", de construire une seconde cascade basée sur l'estimation de l'orientation utilisant le paradigme end-to-end (de bout en bout). Le dernier chapitre sera l'occasion d'explorer des pistes prometteuses dans la création et l'utilisation de données synthétiques. Enfin, la conclusion aura la lourde tâche d'essayer de mettre de l'ordre dans les conclusions des différentes parties, de répondre aux questions non formulées, et d'ouvrir sur des questions plus générales.

Chapitre 2

Détecteurs à un étage pour l'imagerie aérienne

Ce chapitre s'intéresse aux détecteurs d'objets à un étage (aussi appelés détecteurs "single-shot"). Ces détecteurs sont construits sur le principe de la classification de vignettes par fenêtres glissantes, principe déjà évoqué dans l'introduction. Après un bref état de l'art sur les détecteurs d'objets par fenêtres glissantes, nous en proposerons un adapté à notre tache, basé sur des principes de *deep-learning* : le classifieur proposé sera en effet un réseau de neurones convolutionnel profond. Des expériences dans le domaine de l'imagerie aérienne viendront ensuite valider nos choix et apporter des diagnostics sur la méthode proposée. Enfin nous comparerons notre détecteur aux autres détecteurs à un étage de l'état de l'art, ce qui nous permettra d'identifier les éléments importants d'une chaîne de détection et de proposer des améliorations.

2.1 Bref état de l'art sur les détecteurs d'objets récents à un étage

Nous avons expliqué dans la section 1.1.2 le principe des détecteurs par fenêtres glissantes, détecteurs reposant sur des classifieurs appliqués aux vignettes de l'image. Les détecteurs modernes apprennent ces classifieurs sur des BDD au moyen de méthodes d'"apprentissage machine". Les détecteurs que l'on rencontre dans la littérature diffèrent

principalement sur le type de classifieur utilisé.

Si le classifieur fait intervenir des opérations mathématiques lourdes, il est clair que le parcours de l'image peut être trop long pour des applications temps réel. Cela explique une partie de la popularité du classifieur SVM linéaire [36] dont le score est calculé à partir d'un simple produit scalaire sur des descripteurs extraits de la vignette à classer (représentation plus haut-niveau de la vignette).

Malheureusement, qui dit classifieur linéaire dit données linéairement séparables. Or il est connu que la répartition des images naturelles dans l'espace des pixels n'obéit à aucune règle simple : par exemple elles n'occupent pas tout l'espace de manière dense (le bruit blanc n'est pas une image naturelle) et passer d'une image d'un même objet à une autre en utilisant la distance euclidienne produit des images non observables naturellement. L'ensemble est donc non convexe et de structure complexe. Ceci rend difficile la discrimination correcte des classes fines dans cet espace à l'aide d'hyperplans.

Pour nous en convaincre, nous réalisons, sur les conseils d'Andrej Karpathy [124], l'entraînement d'un SVM linéaire sur des vignettes voitures et non voitures (fonds) issues de la BDD MunichDLR3K. Nous prenons toutes les voitures disponibles et les orientons verticalement et prenons un nombre égal de fonds échantillonnés aléatoirement. Il est aisé de voir sur la Figure 2.1 que l'entraînement aboutit à la formation d'un modèle moyen ("template") insuffisant pour des applications réelles (par exemple, imaginer une voiture d'une taille variable). De plus, si nous n'avions pas rectifié préalablement les orientations, le template aurait ressemblé à une moyenne de différents templates à différentes orientations et n'aurait sûrement pas été exploitable.

L'application répétée du SVM sur chaque fenêtre est à mettre en parallèle avec l'un des premiers algorithmes de détection utilisant des techniques de "template matching" [17] dans lesquels il s'agit de trouver le maximum de la carte de corrélation obtenue par l'application d'un template fixe.

Il faut donc intercaler entre la vignette et le classifieur une *transformation non linéaire* permettant de rendre la répartition des nouvelles données (nous utiliserons l'anglicisme "features" ou le mot français descripteurs) plus structurée et donc plus facilement classifiable.

FIGURE 2.1 – De gauche à droite : des exemples de fonds de la base Munich, des exemples de voitures et les poids d'un SVM linéaire entraîné en classification (réorganisés spatialement et normalisés)

C'est cette idée majoritairement qui sera développée dans les années 2000 à travers une multitude de méthodes différentes.

Les idées les plus simples comme l'utilisation de SVMs à noyaux non linéaires (polynomial ou gaussien) sur les pixels ou même l'ajout de degrés de liberté supplémentaires dans le template matching [37, 35] ont vite montré leurs limites.

Pour dépasser les performances, il a fallu s'éloigner de plus en plus des pixels en cherchant des invariants de plus haut niveau.

Les histogrammes de couleur [224], de champs réceptifs [207], les ondelettes de Haar utilisées comme features d'entrée de SVM linéaires ou non linéaires [208, 239, 173] ainsi que les descripteurs de gradients de Gavrila [65] firent leurs preuves, mais furent vite remplacés par les histogrammes de gradients orientés ou HOGs de Dalal et Triggs [43].

Ce qui fait qu'une approche en détection d'objets sera adoptée majoritairement par la communauté ou non repose sur quatre points :

— performance de la méthode en détection sur des benchmarks connus,

— temps de calcul,

— implémentation open source disponible,

— popularité du chercheur/laboratoire à l'origine de la méthode.

Les HOGs [43] et particulièrement le "Deformable Part Model"[56], permettant la

découverte non supervisée de parties d'objets grâce à l'introduction de variables latentes, vérifiant la plupart de ces critères, furent donc massivement utilisés par la communauté.

Entre 2005 et 2012, la très grande majorité des approches de détection classique utilise donc des variantes des HOGs normalisés différemment ou bien combinés avec du bootstrapping et des parties déformables [56]. C'est d'ailleurs directement dans cette lignée que s'inscrivent les premiers travaux sur la base VeDAI de Razakarivony et Jurie [188, 187]. Même si les HOGs sont maintenant quasi abandonnés, la dernière itération du DPM, le DPMv5 [201] très rapide est quelquefois encore utilisé comme baseline compte tenu de sa simplicité d'utilisation ainsi que de sa popularité.

Parmi les alternatives aux HOGs disponibles à l'époque, nous citerons les vecteurs de Fisher FV [178, 203] permettant une réduction de dimensionnalité des descripteurs ainsi que les descripteurs SIFT de David Lowe [154] et leurs normalisations respectives comme [7], qui sont cependant plus utilisés pour la classification, le matching d'images ou la recherche d'instances.

Disposant d'une armée de descripteurs, il est possible d'apprendre à les combiner de manière optimale pour renforcer leur pouvoir discriminatif comme dans [236, 238]. Un seul descripteur peut aussi donner lieu à une multitude de classifieurs linéaires (1 par objet présent dans l'ensemble d'apprentissage) comme dans les Exemplar SVMs de Malisiewicz [158].

En 2012, c'est l'avènement des descripteurs issus de l'entraînement en classification de réseaux de neurones convolutionnels (CNNs) de Yann LeCun[139] grâce au code pour carte graphique de l'équipe SuperVision menée par Alex Khrizhevsky [134] leur ayant permis de gagner avec une marge impressionnante le challenge ImageNet[44]. Les itérations suivantes de ce challenge [257, 226, 90, 225, 256, 106] ont fini d'asseoir la suprématie des CNNs en classification et détection d'objets comme le montre la Figure 2.2. Qui plus est, l'entraînement préalable sur cette grande base de données de ces CNNs permet d'obtenir des descripteurs souvent bien plus généraux et transférables que tous les précédents descripteurs faits main [190]. Ceci est corrélé avec le fait que les "features" apprises par ces méthodes semblent coïncider remarquablement avec notre perception visuelle comme le montre l'étude récente [262].

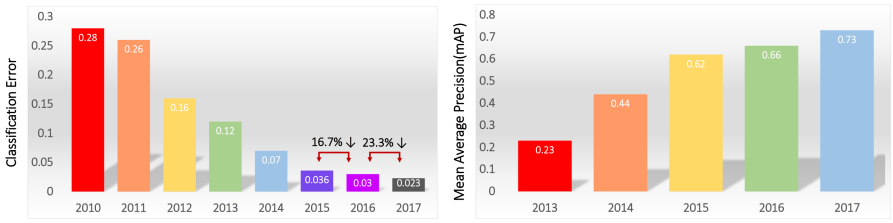


FIGURE 2.2 – Évolution des performances sur Imagenet : à gauche en classification, et à droite en détection. (tiré du workshop ImageNet à CVPR2017)

Depuis cette date, et de plus en plus, force est de constater que, en ce qui concerne la détection d'objets (voir l'entièreté de la vision par ordinateur, mais ce point est débattable), les alternatives se font rares. Nous sommes dans l'ère du "Deep Learning".

Compte tenu de cette remarque et comme la thèse a commencé en 2016, nous centrerons l'intégralité de la discussion sur les méthodes de détection *à base de CNNs*. Malgré tout, à chaque fois que ce sera possible, nous invoquerons ces références pour mettre en lumière des comparaisons ou des parallèles intéressants. Nous gardons aussi l'oeil ouvert pour distinguer un changement de paradigme similaire à ceux de 2005 ou de 2012, mais il semble qu'il n'interviendra pas tout de suite. Le lecteur non familier avec les CNNs est encouragé à feuilleter l'annexe A : Anatomie des CNNs avant de commencer à lire la suite.

Les méthodes à un étage issues du Deep-Learning sont nombreuses. Étant donnée la simplicité de ce type de méthode, nous ne perdrons pas en généralité en centrant la discussion autour des deux plus connues : YOLOv1 [192] et SSD [150]. Elles seront décrites en détail dans la dernière partie de ce chapitre. Le lecteur voulant découvrir d'autres exemples de détecteurs "single-shot" est invité à regarder notre enquête très complète sur ces questions [3].

2.2 Vers un détecteur simple à un étage, adapté à l'imagerie aérienne

Cette section propose un détecteur simple adapté à l'imagerie aérienne. Il repose sur un classifieur CNN (descripteur + classifieur appris simultanément) appliqué par

Nom	Type	Taille du filtre Pas	Taille de l'entrée	Taille de la sortie
Conv1	convolution	5x5/1	45x45x3	41x41x96
Pool1	max-pooling	2x2/2	41x41x96	20x20x96
Conv2	convolution	5x5/1	20x20x96	16x16x192
Pool2	max-pooling	2x2/2	16x16x192	8x8x192
fc1	fully connected	8x8/1	8x8x192	1x1x384
fc2	fully connected	1x1/1	1x1x384	1x1x84
fc3	fully connected	1x1/1	1x1x84	1x1x2
Softmax	Softmax	None	1x1x2	1x1x2

TABLE 2.1 – Architecture de notre réseau, inspiré par LeNet-5

fenêtre glissante sur les images. Nous décrivons d'abord les composants de ce CNN ainsi que les détails de son entraînement. Nous conduisons ensuite des expériences avec ce détecteur sur une BDD d'imagerie aérienne : VeDAI [189], que nous avions déjà introduite section 1.2.2.

2.2.1 Méthodologie

Corps du réseau

Nous souhaitons ici construire le détecteur le plus simple possible, avec l'objectif de pouvoir l'entraîner "from scratch" c'est-à-dire depuis un CNN initialisé aléatoirement (cf. Annexe A pour des commentaires sur ce que cela signifie en termes d'optimisation). Ces choix sont motivés par la différence forte entre les images aériennes et les bases publiques disponibles pour pré entraîner les modèles. De plus, les vignettes à classer étant de petite taille et les objets ne changeant pas de point de vue, utiliser un classifieur plus simple que ceux utilisés pour la détection des objets dans des bases de type MS COCO nous paraît préférable.

Nous utilisons une des structures CNN les plus simples, le réseau LeNet-5 [139] à 4 couches cachées avec filtres 5x5 que nous initialisons en utilisant une des heuristiques récentes (voir Annexe A) afin de faciliter l'optimisation. Le corps du réseau est représenté Table 2.1. Les paramètres tels que la taille des vignettes d'entrée (45 par 45) le nombre de descripteurs "carte" (nous reviendrons sur la raison de cette appelation plus tard dans cette section) par couche et la taille des filtres sont choisis par validation croisée sur la base étudiée.

Inférence

Parcours par fenêtre glissante et NMS. Nous parcourons toutes les images à une seule résolution par fenêtre glissante avec un pas de p pixels. L'application du classifieur sur l'image entière donne donc lieu à une grille de score. Le maillage de la grille est défini par le pas de la fenêtre glissante multiplié par la résolution de la sous-image de la pyramide sur laquelle elle est appliquée. Cela provoque un problème : nous pouvons en effet apercevoir des zones de points chauds étalés (blobs) correspondant à différentes fenêtres plus ou moins centrées sur l'objet et donc ayant toutes des scores élevés. En repassant dans le plan image, cela donnerait lieu à beaucoup de détections du même objet, donc à une faible précision.

Pour éliminer les détections redondantes, nous pouvons utiliser une étape dite de NMS (suppression des non-maximas).

L'heuristique utilisée depuis des années consiste à prendre le maximum global de la grille puis à assigner un score nul à son voisinage proche et à réitérer le processus jusqu'à obtenir un isolement de tous les maximas. Bien sûr, *si le seuil est trop grand, nous perdons en rappel. S'il est trop petit, nous perdons en précision.*

Cette procédure empirique introduit son lot de paramètres supplémentaires (taille du voisinage, prise en compte de la taille des fenêtres).

Récemment, il a été découvert qu'appliquer une baisse des scores continue des voisins plutôt qu'une affectation dure (à -inf) permettait de gagner plusieurs points de mAP sur une large gamme de détecteurs [14]. Il est même possible d'apprendre cette étape afin d'éviter l'arbitraire [105].

Accélération du système en inférence : le tout convolutionnel. Le passage par fenêtre glissante pour le test du détecteur après son apprentissage est relativement long même pour un réseau comme LeNet-5 comportant beaucoup moins de couches que les réseaux plus récents. Pour gagner du temps de calcul, il est possible d'utiliser l'astuce du tout-convolutionnel. Lorsque nous faisons glisser la fenêtre de quelques pixels et que nous recalculons le descripteur convolutionnel (avant les couches complètement connectées) sur une autre fenêtre adjacente recouvrant en partie la première, la plupart des calculs

ont déjà été effectués il est donc inutile de les refaire. Pour comprendre ce qui se passe, nous considérons l'exemple de deux vignettes adjacentes I_1 et I_2 de même taille W et H avec I_1 à gauche de I_2 et un recouvrement de Δ_x pixels. Nous cherchons à calculer les descripteurs correspondants à la première couche de convolutions d'un CNN relatifs à ces vignettes D_1 et D_2 dans le cas de poids constants.

$$D_1 = \sum_{\substack{0 \leq x \leq W \\ 0 \leq y \leq H}} activation(\langle\ w_c, I_1(x,y) \rangle) \tag{2.1}$$

$$D_2 = \sum_{\substack{0 \leq x \leq W \\ 0 \leq y \leq H}} activation(\langle\ w_c, I_2(x,y) \rangle) \tag{2.2}$$

En considérant les deux vignettes comme une même grande vignette I, (ce qui est possible si le filtre divise la taille de la première vignette mais est une approximation dans le cas général) nous pouvons écrire :

$$D = \sum_{\substack{0 \leq x \leq 2W - \Delta_x \\ 0 \leq y \leq H}} activation(\langle\ w_c, I(x,y) \rangle) \tag{2.3}$$

Et nous retrouvons

$$\begin{aligned} D_1 &= \{d(x,y) \in D | (0 \leq x \leq W)\} \\ D_2 &= \{d(x,y) \in D | (W - \Delta_x \leq x \leq 2W - \Delta_x)\} \end{aligned} \tag{2.4}$$

De proche en proche il est possible d'obtenir tous les descripteurs associés aux différentes vignettes possibles en coupant au même endroit le réseau appliqué *à l'image entière.*

Cette abstraction semble ne pas apporter grand-chose à première vue, mais les convolutions étant très facilement parallélisables sur GPU, nous venons de faire un bond énorme en rapidité.

Comme la structure spatiale de l'image reste contenue dans cette première couche du descripteur, nous parlons de "descripteur carte" (*feature map* en anglais).

Les CNNs cependant comportent aussi des couches de max-pooling. Ces statistiques locales (max-pooling ou average pooling en général) permettent de condenser l'information et de rendre le descripteur invariant aux petites déformations locales. Il y a

bien évidemment un parallèle à faire avec les cellules des HOGs. Le plus utilisé dans la littérature (quasi hégémonique) étant le max-pooling 2x2 non recouvrant, qui revient à appliquer l'opérateur maximum sur des cellules spatiales carrées de 4 pixels (voir Annexe A).

En appliquant cette opération sur la carte convolutionnelle, en la considérant comme une grande vignette (comme pour les premières convolutions), il est facile de voir qu'en coupant au bon endroit la carte obtenue, nous retrouvons les descripteurs voulus, *mais cette fois-ci seulement pour les vignettes étant séparées d'un nombre pair de pixels de la première*. Nous avons perdu tous les descripteurs impairs : la résolution de la carte a donc été divisée par 2. Ce n'est pas très limitant, car de toute façon dans les chaînes de détection de l'état de l'art il est souvent choisi d'ignorer des positions de fenêtres en choisissant un pas de fenêtre supérieur à 1 pour gagner en rapidité (*tout en perdant malgré tout en précision de localisation et en rappel*). Il s'agit même, d'une certaine manière, d'un NMS caché : nous avons pris le maximum local et éteint les voisins.

Il est possible de continuer l'application de convolutions et de max-pooling, les deux opérations composant le corps de tous les CNNs, autant de fois que le permet la résolution voulue. À chaque application de max-pooling, nous multiplions le pas de la fenêtre glissante équivalente par deux. Si nous voulons une sortie plus résolue, il suffit d'appliquer le réseau à des versions décalées de l'image entière [210].

Il est intéressant d'introduire ici la notion de *champ réceptif*, qui sera d'une importance capitale pour toute la suite. Il s'agit pour un pixel d'un descripteur carte d'un niveau donné de *l'ensemble des pixels de l'image de départ intervenant dans son calcul*. Nous reviendrons sur la raison pour laquelle cette définition est cruciale dans le design d'une architecture. Nous donnons ici un exemple de calcul de champ réceptif qui nous donne la taille de la vignette équivalente. Le champ réceptif r suit la relation de récurrence suivante (cette relation peut aussi se trouver en backpropagant les gradients par rapport aux pixels de l'entrée [98]) :

$$\begin{aligned} r_{out} &= r_{in} + (k-1) * j_{in} \\ j_{out} &= j_{in} * s \end{aligned} \tag{2.5}$$

Avec r_{in} le champ réceptif en entrée, r_{out} en sortie, k la taille du filtre ou du pooling associé à cette zone (nous considérons que tous les pixels d'une zone de pooling interviennent dans le calcul), j le saut de pixels en entrée et en sortie avec les mêmes conventions et s le pas du filtre. Pour notre réseau LeNet-5, nous trouvons successivement en initialisant r et j à 1 et en suivant la Table 2.1 :

— $r_1 = 5$, $j_1 = 1$

— $r_2 = 6$, $j_2 = 2$

— $r_3 = 14$, $j_3 = 2$

— $r_4 = 16$, $j_4 = 4$

— $r_5 = 44$, $j_5 = 4$

— $r_f = 44$, $j_f = 4$

Nous retrouvons à un pixel près d'effet de bord dû au pooling la taille de notre vignette d'entrée lorsque nous entraînions en classification.

Un CNN classique appliqué à une image de taille fixe (227x227 pour IMAGENET) comporte après ses couches de convolution/pooling des couches dites complètement connectées prenant des entrées de taille fixe. À première vue, il est donc difficile d'utiliser la même astuce, car maintenant la taille est fonction de celle de l'image.

Dans AlexNet [135] par exemple nous obtenons en sortie des convolutions et poolings un descripteur de dimension (256,6,6) qui est ensuite projeté sur un espace de dimension 4096. La matrice de projection (9216,4096) est donc inutilisable telle quelle si le descripteur est maintenant 256*20*20 par exemple.

Mais ces couches complètement connectées peuvent tout à fait être implémentées comme des convolutions dégénérées. Il suffit de convoluer la matrice de projection spatialement sur le descripteur, peu importe sa taille.

L'application classique d'un CNN sur une image entière permet donc d'obtenir la carte des scores que nous aurions obtenue par parcours de fenêtres glissantes avec un pas équivalent au double du nombre de couches de max-pooling, modulo quelques effets de bord induits par la taille des filtres. C'est la même chose si nous voulons appliquer un SVM linéaire sur un descripteur CNN.

Cette quasi-équivalence popularisée par OverFeat [210] est utilisée maintenant dans

l'intégralité des chaînes de détection à base de CNNs.

Entraînement du classifieur

Déséquilibre des classes. Lorsque l'on entraîne un classifieur utilisé dans un but de détection d'objets, il y a souvent un déséquilibre des classes, en particulier entre la (ou les) classe(s) cible(s) et les fonds. Pour éviter que le classifieur n'apprenne la fonction triviale assignant à toutes les fenêtres un score faible ou nul nous sommes obligés de contrôler ce déséquilibre des classes.

Pour les fonds, nous proposons de réaliser un sous-échantillonnage aléatoire des fonds (majority undersampling [24]) afin de contrôler ce déséquilibre de classes. Nous utilisons d'ailleurs un coût de distance de Kullback-Leibler entre la distribution des labels et celle des sorties du réseau (voir Annexe A), que nous recalibrons pour mettre plus d'importance sur le gradient des exemples positifs mal classés. Pour cela nous calculons leur proportion respective dans l'ensemble d'entraînement courant et utilisons le ratio de déséquilibre pour pondérer chaque exemple :

$$C_W(\mathcal{X}) = -\sum_{i=1}^{N} y_i * log(f_W(x_i)) * R(y_i) \tag{2.6}$$

Avec $R(x) = \frac{N_-}{N_+}$ si $x = 1$, 1 sinon et respectivement N_- et N_+ la proportion d'exemples négatifs et positifs dans l'ensemble de vignettes d'apprentissage, y_i est le vecteur étiquette (en convention "one-hot") associé à l'exemple x_i et f_W la sortie normalisée du réseau. Les alternatives à cette approche de recalibrage d'importance consistent le plus souvent à faire du sur-échantillonnage de la classe minoritaire. L'algorithme le plus connu SMOTE [24] rajoute des positifs synthétiques en interpolant entre les positifs existants. Il ne peut pas s'appliquer pour des images compte tenu des remarques précédentes. C'est le cas aussi de ses extensions comme ADASYN [86] ou SMOTE-boost [25].

Ainsi nous ne pouvons pas utiliser tous les fonds disponibles en une seule fois. Ce sujet du choix des exemples pour l'apprentissage est d'importance capitale en Deep Learning et en apprentissage supervisé en général. Par exemple, les exemples positifs peuvent présenter des cas non typiques dus aux prises de vues, qui peuvent induire en erreur le

classifieur comparé à d'autres exemples plus caractéristiques. Ceci est étudié dans [137]. De plus lorsqu'il y a beaucoup d'images à annoter il arrive qu'il y ait même des erreurs de labellisation (c'est le cas dans la base VeDAI [189]). Bien entendu, les identifier pour les enlever de l'ensemble d'apprentissage comme le fait [5] est souvent primordial. Comme précédemment, il est intéressant de faire une analogie avec RANSAC [58], méthode itérative très utilisée pour la régression robuste et l'identification des données aberrantes.

Sélection des exemples de fond. Il est hors de question de collecter tous les fonds pour des questions de mémoire et à cause du déséquilibre de classes (voir paragraphe précédent). *Il s'agit donc de choisir les exemples qui rentrent dans notre ensemble d'apprentissage.*

Il existe différentes catégories d'exemples négatifs. Ceux mal classés donnent donc accès à un gradient fort et orientant et ceux déjà bien classés donnent lieu à de très faibles gradients.

Supposons que la sortie s avant le softmax de notre réseau $f_W(x)$ soit de taille C (avec C le nombre de classes) et que nous utilisons le coût de la distance de Kullback-Leibler classique CE. En posant x_i une vignette, y_i le vecteur de classe associé encodé en "one-hot" (vecteur nul de taille C, sauf en c_{gt} où il vaut 1) et W les poids du réseau, la règle de la chaîne permet de dérouler le calcul du gradient du coût par rapport à W

(de taille le nombre de poids) :

$$
\begin{aligned}
CE(x_i, W) &= -\langle log(f_W(x_i)), y_i \rangle \\
\nabla_W CE(x_i, W) &= -J_{log(f_W(x_i))}(W) * y_i \\
J_{log(f_W(x_i))}(W)(j,c) &= -\frac{\partial log(softmax(s))_c}{\partial W_j} = -\sum_{c'=1}^{C} \frac{\partial softmax(s)_c}{\partial s_{c'}} \frac{\partial s_{c'}}{\partial W_j} \frac{1}{softmax(s)_c} \\
-\frac{\partial softmax(s_c)}{\partial s_{c'}} &= -\frac{\frac{\partial exp(s_c)}{s_{c'}} * \sum_{k=1}^{C} exp(s_k) - exp(s_c) * exp(s_{c'})}{(\sum_{k=1}^{C} exp(s_k))^2} \\
-\frac{\partial softmax(s_c)}{\partial s_{c'}} &= -softmax(s)_c(\delta_{cc'} - softmax(s)_{c'}) \\
J_{log(f_W(x_i))}(W)(j,c) &= \sum_{c'=1}^{C} (softmax(s)_{c'} - \delta_{cc'}) \frac{\partial s_{c'}}{\partial W_j} \\
\nabla_W CE(x_i, W) &= \sum_{c'=1}^{C} (softmax(s)_{c'} - \delta_{c_{gt}c'}) \frac{\partial s_{c'}}{\partial W_j}
\end{aligned}
\tag{2.7}
$$

Ou $\delta_{cc'}$ vaut 1 si $c = c'$, 0 sinon. Il y aura donc toujours du gradient que ce soit pour rapprocher *directement* de 1 la case de la sortie voulue $s_{c_{gt}}$ ou pour baisser les autres faisant monter $s_{c_{gt}}$ grâce à la normalisation. Mais plus $s_{c_{gt}}$ est loin de 1 (donc plus mal l'exemple est classé plus le gradient est fort).

Il faut donc aller chercher les exemples négatifs difficiles à classifier quitte à évincer de l'apprentissage les exemples trop faciles et peu importants en termes de gradients.

Dans le cas d'un SVM [36], la distinction est encore plus flagrante, car les exemples non à côté de la frontière ont un coût nul et un gradient à 0. De plus, le problème étant convexe ils n'ont donc absolument aucune influence sur l'optimisation. Ce processus de sélection des exemples négatifs difficiles est appelé dans la littérature : "hard-examples mining" ou "bootstrapping" (parfois abrégé hard-mining HM). Il a été proposé notamment dans le DPM [56], où il est démontré la convergence de l'optimisation par passes de bootstrapping successives, et est toujours très utilisé, notamment dans le récent Online Hard Examples Mining (OHEM) [215].

Nous réalisons 4 différentes stratégies de hard mining et regardons les bénéfices associés : le bootstrapping offre de nombreuses libertés sur la manière dont les exemples

difficiles sont choisis. Nous pourrions par exemple choisir un nombre limité de faux positifs par image, ou nous pourrions fixer un seuil de score et ne collecter un faux positif que si son score est supérieur à ce seuil (0.5 semble être un choix raisonnable lorsque le score est normalisé). Il y a aussi la question des faibles vrais positifs : s'il faut vraiment les inclure et si oui à partir de quel score ? Toutes nos premières expériences utilisaient les seuils (0.5,0.5) tout en limitant le nombre d'exemples difficiles par image à 25. Nous testons 4 variantes :

— Stratégie 1 : nous choissons 25 faux positifs par image quels que soient leurs scores sans prendre aucun vrai positif de score faible.
— Stratégie 2 : Même stratégie que 1, mais en rajoutant les vrais positifs faibles dont le score est inférieur à 0.5.
— Stratégie 3 : nous choisissons tous les vrais positifs dont le score est inférieur à 0.5 et les faux positifs dont le score est supérieur à 0.5.
— Stratégie 4 : Comme la stratégie 3, sans les vrais positifs faibles.

Entraînement tout convolutionnel. Les deux paragraphes précédents évoquent des problèmes liés à l'apprentissage par vignettes en classification. Comme la classe de chacune des vignettes obtenue en parcours par fenêtre glissante est connue (en fonction de notre définition de la détection) nous connaissons la carte vérité terrain. En utilisant la même transformation du réseau évoquée pour l'inférence, il est donc possible d'utiliser un softmax à 2 dimensions à l'entraînement possédant la cible du softmax la carte "vraie". Il s'agit peu ou prou de ce que fait Shelhamer [211] pour de la segmentation sémantique sauf qu'il y a ici un facteur de sous-échantillonnage et donc nous n'obtenons pas une classification pour tous les pixels. Nous pouvons donc entraîner notre réseau à la manière d'un entrainement en segmentation sémantique et le tester en détection, comme le montre la Figure 2.3. En choisissant ce mode de fonctionnement nous pouvons utiliser toutes les heuristiques développées dans la littérature de segmentation sémantique. L'entraînement tout convolutionnel pose cependant des problèmes. Là il n'est soudain plus possible à première vue de contrôler la proportion classe minoritaire/classe majoritaire. Cependant c'est une impression fausse : nous pouvons faire un échantillonnage de la

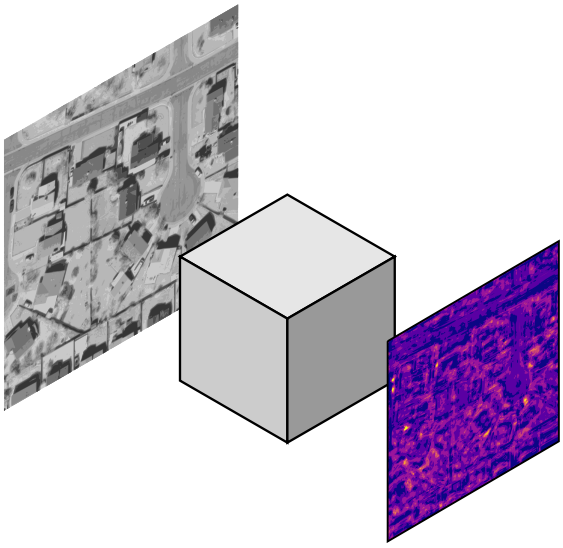


FIGURE 2.3 – Forward d'une image de VeDAI par notre réseau LeNet-5 produisant une carte de score

carte et ne back-propager le gradient que dans les indices visés ou bien utiliser comme précédemment des poids différents sur les positifs et négatifs. C'est le choix de HED [251], un détecteur de contours complètement convolutionnel, les contours ne présentant que, en général, 10% de la surface de l'image sont analogues à nos cibles qui ne couvrent qu'une partie infime de la carte. Sa stratégie à lui est "en ligne" c'est-à-dire que les poids changent en fonction de la proportion courante de contours sur l'image dans le mini-batch. Nous pouvons aussi citer la fonction de coût DICE [162], qui permet de se défaire de poids.

$$DICE(y(k), f_w(x_k)) = \frac{2 * \sum_{i=1}^{N} y(k)_i f_W(x_k)_i}{\sum_{i=1}^{N} y(k)_i^2 + \sum_{i=1}^{N} f_W(x_k)_i^2} \tag{2.8}$$

Il est important de comprendre les possibilités incroyables de cette formulation complètement convolutionnelle. Le chercheur dispose maintenant d'un régresseur par pixel, qui peut être complètement général (probabilité, scalaires, etc.) et que nous pouvons entraîner d'un bout à l'autre (end-to-end) sans avoir besoin de faire les choses par étapes (entraînement sur vignettes, parcours par fenêtre glissante, etc.) *à supposer que toutes les opérations utilisées soient différentiables ou sous-différentiables.*

2.2.2 Validation expérimentale du détecteur proposé

La base d'image utilisée pour les expériences

Pour évaluer notre détecteur, nous nous sommes concentrés dans un premier temps sur la base VeDAI [189] évoquée précédemment et particulièrement sur la classe voiture. Les images choisies sont les Petites Images IR 512x512 (PII). Nous sur-échantillonnons ces images d'un facteur 2 par interpolation bilinéaire, de manière à compenser le sous-échantillonnage réalisé par les couches des réseaux de neurones qui vont traiter ces images. Les métriques de performance, qui nous intéressent sont les courbes de Précision-Rappel, l'Average Precision AP et la mAP moyennée sur les 10 folds. La détection est au sens de VeDAI comme expliqué dans le chapitre précédent.

Détection de la classe voiture sur VeDAI

Sans Hard-negative Mining. Nous apprenons un classifieur sur une base de vignettes composées de voitures et de fonds, comme pour le SVM linéaire appris sur la base MunichDRL3K, mais nous ne normalisons pas les angles des voitures.

Nous choisissons une taille de fenêtre de 45 par 45 pixels afin d'avoir l'intégralité de la voiture et quelques pixels de fond dans le patch, le pas de la fenêtre glissante est de 4 pixels. Il est facile de choisir les voitures, car elles sont disponibles en nombre limité dans la base d'entraînement. Comme pour Munich nous montrons un échantillon des vignettes utilisées pour l'entraînement, Figure 2.4.

Nous entraînons notre réseau comme expliqué précédemment sans utiliser de boots-trapping (entraînement sur vignettes, inférence par fenêtre glissante classique donc non complètement convolutionnel). Lorsque nous calculons la MAP sur 11 points de notre détecteur et que nous le comparons à l'état de l'art table 2.2, il reste du chemin à faire. Nous testons aussi l'inférence toute convolutionnelle : compte tenu de ces deux couches de pooling, le pas équivalent de la fenêtre glissante est donc de 4. Les performances obtenues sont sensiblement les mêmes de l'ordre de 30% (26.52 ± 4.1), mais maintenant l'inférence est beaucoup plus rapide : nous passons de quelques heures d'inférence

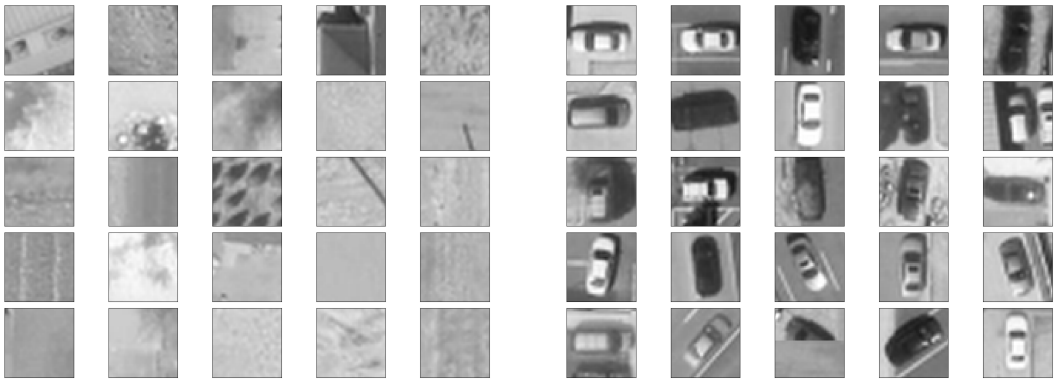


FIGURE 2.4 – Des exemples des fonds VeDAI sont à gauche et des positifs à droite (classe voiture)

methode	mAP	Rappel@0.01FPPI
DPM [189]	60.5 ± 4.2	13.4 ± 6.8
SVM+HOG31 [189]	55.4 ± 2.6	7.8 ± 5.5
SVM+LBP [189]	51.7 ± 5.2	5.5 ± 2.2
SVM+LTP [189]	60.4 ± 4.0	9.3 ± 3.7
SVM+HOG31+LBP [189]	61.3 ± 3.9	8.3 ± 5.2
SVM Fusion AED (HOG) [187]	$\mathbf{69.6 \pm 3.4}$	$\mathbf{20.4 \pm 6.2}$
Wide LENET-5 ("from scratch")	26.37 ± 3.77	$0. \pm 0.$

TABLE 2.2 – Comparaison de différents algorithmes de détection sur les 10 folds de VeDAI

à quelques minutes pour une centaine d'images, nous avons gagné un facteur de 50.

Avec Hard-negatif mining. Nous avons réussi à accélérer notre chaîne, mais nous avons toujours le problème de sa performance de détection. Des expériences où l'ensemble des images entières d'où sont issues les vignettes est utilisé pour tester l'algorithme montrent que, même sur cet ensemble d'images, les performances du réseau sont faibles : il y a beaucoup trop de faux positifs. Ainsi, le réseau étant très peu performant sur les images d'apprentissage, il y a peu de chance pour qu'il généralise sur d'autres images (considérer l'exemple de régression d'un sinus au chapitre précédent). En général les deux raisons invoquées pour expliquer le sous-apprentissage sont :

— la capacité du réseau : pas assez de poids (ce qui se comprend si la frontière entre les exemples des deux classes est arbitrairement complexe)

— des problèmes d'optimisation : [91] par exemple remarque que lorsque nous ajou-

Strategy	R1	R2	R3	R4	R5	R6
S1	0.00	0.21	0.39	0.36	0.28	0.25
S2	0.00	0.10	0.17	0.43	0.27	0.32
S3	0.00	0.35	0.34	0.31	0.25	0.29
S4	0.00	0.23	0.32	0.23	0.32	0.32

TABLE 2.3 – RAPPEL@0.01FPPI en fonction du nombre de passes de hard-mining pour les différentes stratégies

tons trop de couches à un CNN régulier la performance en classification sur l'apprentissage baisse, signe que l'optimisation se passe mal. C'est la raison pour laquelle ils ont introduit les connexions skips (ou saute-mouton)(voir Annexe A).

Ces deux raisons sont assez vite évacuées, car ajouter des filtres au réseau ne change rien et la performance en classification sur l'ensemble d'apprentissage *des vignettes de classification* est parfaite (100% de précision en classification à un seuil de 0.5) ! Le problème semble en fait venir de la présence de fond difficiles, jamais vu dans l'entraînement. C'est pour cela que nous introduisons les différentes stratégies de HM présentées section précédente. Ces 4 stratégies sont évaluées table 2.3 (à une échelle $\frac{1}{100}$ de la table 2.2). La meilleure stratégie est donc S2 en termes de rappel moyen. Mais si l'utilisateur veut limiter le nombre de passes, S3 doit être favorisée, car elle donne un bon rappel à 0.01 FPPI après seulement deux étapes. Les performances saturent, voire baissent après un nombre fixé de passes (de 0.28 à 0.25 avec S1 entre la passe 5 et 6 par exemple). En effet, au fur et à mesure que nous collectons les exemples difficiles et les incluons dans l'apprentissage, les performances du détecteur augmentent exponentiellement sur l'ensemble d'apprentissage. Ceci est présenté Figure 2.5 dans laquelle nous montrons l'évolution des courbes précision-rappel sur l'ensemble d'entraînement après différents nombres de passes de hard negative mining.

Il est intéressant de remarquer plusieurs choses :

— La collecte d'exemples est réalisée "offline" c.-à-d. nous entraînons jusqu'à convergence, puis nous récupérons les exemples

— Il est nécessaire de réaliser plusieurs passes pour collecter des exemples de plus

methode	mAP	Rappel@0.01FPPI
DPM [189]	60.5 ± 4.2	13.4 ± 6.8
SVM+HOG31 [189]	55.4 ± 2.6	7.8 ± 5.5
SVM+LBP [189]	51.7 ± 5.2	5.5 ± 2.2
SVM+LTP [189]	60.4 ± 4.0	9.3 ± 3.7
SVM+HOG31+LBP [189]	61.3 ± 3.9	8.3 ± 5.2
SVM Fusion AED (HOG) [187]	69.6 ± 3.4	20.4 ± 6.2
Wide LENET-5 ("from scratch")	26.37 ± 3.77	$0. \pm 0.$
Wide-LENET-5+HM (CV)	$\mathbf{77.80 \pm 3.3}$	31.04 ± 11
Wide-LENET-5+HM (manuel)	75.13 ± 3.25	$\mathbf{33.42 \pm 9.60}$

TABLE 2.4 – Comparaisons avec les algorithmes existants

en plus difficiles, chaque réseau étant meilleur que son prédécesseur.

— La AP ne sature pas à 100, ceci peut s'expliquer par la présence de véhicules en bords d'images qui ne peuvent être détectés par le réseau (si il n'y a pas de precision à rappel 100 l'AP est donc bornée par 10/11=91%)

— Le réseau commence parfois peut-être à sur-apprendre lorsque nous effectuons trop de passes

Nous présentons aussi un échantillon des exemples difficiles collectés c'est-à-dire les erreurs de haut score du détecteur Figure 2.6.

Il est aussi intéressant de s'interroger sur nos conclusions ; c.-à-d. par exemple si la sélection des exemples n'est pas vraiment importante en elle même, mais que seul le nombre de fonds influe véritablement sur les performances finales du réseau. Nous collectons donc au hasard autant de fonds que récoltés après 6 passes de hard-mining, l'écart de performances observé sur la courbe Précision-Rappel de la Figure 2.7 n'est donc explicable que par l'importance de certains exemples difficiles visibles sur la Figure 2.7.

Cela nous donne notre premier résultat état de l'art.

Choisir les paramètres du CNN. Le choix du nombre de filtres utilisés dans chacune des couches de notre LeNet-5 était quelque peu arbitraire. Nous avions en fait effectué préalablement une validation croisée sur les architectures en même temps qu'une étude sur les coefficients de régularisation Ridge (ou "weight decay" dans la littérature de réseaux de neurones) à rajouter au coût pour lisser les filtres (c'est équivalent dans le

cas d'une régression linéaire à un a priori gaussien de moyenne nulle sur les poids [13]). Les résultats de l'étude sont présentés dans la Table 2.5. On rappelle que les différentes colomnes de type (N_1, N_2, N_3, N_4) représentent le nombre de filtre, c'est-à-dire la profondeur des cartes, dans chaque couche du réseau LeNet-5. Les résultats sont donnés après apprentissage sur "train" et test sur "val" moyenné sur 5 découpes (fold 1 à 5).

La table est facilement interprétable. L'architecture (6,16,120,84) sous-apprend complètement et n'est pas souhaitable. C'est aussi le cas dans une moindre mesure des architectures des colomnes 1,2 5 et 6 qui donnent leurs meilleures performances quand elles ne sont pas fortement régularisées. Il faut donc bien des réseaux relativement larges, des Wide-LENET-5 (le préfixe "Wide" est employé lorsque l'architecture (96, 192, 284, 84) est utilisée) régularisé à 0.001 (trop de régularisation donne des résultats mauvais) pour atteindre la crête de la performance. La même architecture avec un seuil de régularisation choisi manuellement ("manuel") donne dans la table 2.4 75.13%± 2.35 de mAP et 33.42%± 9.6 de RAPPEL@0.01FPPI. Nous gagnons donc 3 points de mAP, même si nous perdons un peu en rappel malheureusement (2 points entre 33.42 et 31.04) par rapport à l'architecture choisie suite à l'étude de la table 2.5 nommée "CV" pour validation croisée.

Étude sur le contexte. Lors de la définition de la taille des fenêtres il n'est pas complètement trivial de choisir le bon nombre de pixels de fond à introduire dans une région à classifier. Trop peu de pixels, et aucune information sur le contexte n'est disponible, ce qui rend la tâche ardue lorsque nous travaillons en faible résolution. Mais utiliser trop de pixels introduit trop de bruit et rend compliquée la recherche d'information. Une étude est réalisée pour différentes tailles de fenêtres. Les chaînes de détection récentes se sont aussi posé la question du contexte. Nous fixons l'architecture et nous changeons la taille des patchs prélevés en incluant plus ou moins de contexte. Nous regardons ensuite l'évolution des performances en fonction des passes de hard negative mining.

La figure 2.8 montre que la taille 45x45 est optimale quand la classification est rudimentaire et doit séparer voitures de fonds dans les deux premières passes, mais que plus de contexte est nécessaire lorsque suffisamment d'exemples difficiles ont été collectés

λ/Architecture	$(32, 64, 120, 84)$	$(64, 128, 256, 84)$	$(96, 192, 384, 84)$	$(6, 16, 120, 84)$	$(32, 64, 128, 4096)$	$(32, 64, 128, 32)$
0.0	$65.14 \pm 4.61/25.62 \pm 7.96$	$67.35 \pm 4.70/20.20 \pm 3.89$	$68.02 \pm 5.87/14.05 \pm 11.84$	$42.43 \pm 9.63/7.72 \pm 6.48$	$63.15 \pm 2.48/25.49 \pm 4.15$	$63.74 \pm 4.12/22.79 \pm 7.57$
0.0001	$66.14 \pm 3.72/26.06 \pm 5.37$	$\mathbf{70.18 \pm 3.49/30.65 \pm 6.65}$	$70.72 \pm 3.26/34.07 \pm 1.97$	$40.35 \pm 6.71/5.62 \pm 1.02$	$65.31 \pm 5.50/19.27 \pm 6.96$	$63.84 \pm 5.32/20.96 \pm 5.82$
0.001	$\mathbf{68.12 \pm 3.74/28.30 \pm 6.59}$	$69.85 \pm 3.66/26.73 \pm 8.26$	$\mathbf{73.32 \pm 4.13}$ / $\mathbf{35.27 \pm 7.56}$	$\mathbf{49.19 \pm 10.42/12.58 \pm 4.65}$	$\mathbf{68.30 \pm 6.24/25.76 \pm 11.20}$	$\mathbf{68.03 \pm 5.66/29.40 \pm 7.12}$
0.01	$64.41 \pm 12.42/19.10 \pm 7.90$	67.07 ± 9.73 16.12 ± 9.28	63.74 ± 12.73 / 17.62 ± 8.60	46.98 ± 12.49 / 8.21 ± 2.41	63.37 ± 10.99 / 13.64 ± 6.41	64.88 ± 10.41 / 18.23 ± 9.34

TABLE 2.5 – Table représentant la mAP (sur 5 folds) et le RAPPEL@0.01FPPI moyen en fonction des architectures et de la régularisation L2 employée

pour discriminer finement les voitures des exemples très difficiles. Il est aussi intéressant de voir que la taille 61x61 est bien meilleure que les autres après plusieurs passes et que les performances se dégradent d'autant plus au delà de 61x61 pixels avec le contexte lorsque nous ajoutons plus de pixels.

Étude de la normalisation des vignettes pendant l'apprentissage sur les performances en classification

Après avoir observé l'effet des différents paramètres sur la tâche de détection nous nous focalisons sur la partie classification pour étudier plus finement l'influence de la normalisation des vignettes sur la classification. Dans ce but nous construisons plusieurs ensembles de classification. Nous avons regardé toutes les combinaisons possibles de : normaliser les patchs en contraste, en angle, de prendre des positifs décalés de 4 pixels en distance échiquier. Pour plus de simplicité nous avons adopté la notation suivante : S et $\not{S}$ veulent dire respectivement avec normalisation de position (toutes les vignettes sont centrées en les véhicules) et sans : nous déplaçons les vignettes de quelques pixels autour de leur centre, A et $\not{A}$ avec normalisation d'angle ou sans, et, C et $\not{C}$ avec normalisation de contraste ou sans. Les différents résultats visibles table 2.6, montrent l'énorme gain de performance obtenu allant quasi du simple au double lorsque les objets cibles sont tous normalisés au vrai angle. Ce n'est pas vraiment surprenant d'ailleurs, car *avec cette normalisation les différences intra-classes diminuent et il devient donc plus simple de faire de la discrimination fine. Plus il y a de normalisations, meilleure est la performance. Il faut donc pour une classification optimale des cibles bien centrées et localisées au bon angle.*

Il est à noter que ces différentes normalisations sont difficiles à effectuer de manière complètement convolutionnelle. Cela constitue une piste de recherche intéressante. Par exemple, prédire une valeur d'orientation et de localisation pour chaque pixel d'un descripteur carte est possible, mais tourner ces voisinages de pixels est difficile, car mal défini si les voisinages se touchent. C'est cependant ce qu'envisage [31], qui utilise des modules "spatial transformers" [118] complètement convolutionels, idée que nous avions eu simultanément et qui a donc été abandonnée. L'astuce consiste à se débarrasser du

Config.			Préc.	Préc. pos.	Préc. fonds	Rappel à 0.001 FP
~~A~~	~~S~~	~~C~~	98.35	53.91	98.94	28.02
		C	98.94	43.21	99.68	25.59
	S	~~C~~	98.78	66.67	99.21	41.86
		C	99.01	66.67	99.44	50.33
A	~~S~~	~~C~~	99.01	82.72	99.23	63.45
		C	99.10	86.42	99.27	71.68
	S	~~C~~	99.33	91.77	99.43	82.79
		C	**99.41**	**92.59**	**99.50**	**86.58**

TABLE 2.6 – Effets des différentes normalisations sur les performances en Précision en Classification

recouvrement des patchs en créant une carte SS fois la taille de la carte sur laquelle ils sont appliqués avec $SS = S * S$ et S la dimension du patch transformé par le module. Ceci est représenté sur la Figure 2.9.

Étude sur la confusion entre les classes

L'entraînement du détecteur Single-Shot permet de voir les difficultés qu'ont les détecteurs à un étage à faire de la discrimination fine entre les classes. Ceci est visible en regardant l'évolution des classifications correctes des autres classes comme étant des fonds (comprendre non voitures) en fonction des passes de hard-negative mining Figure 2.10. Ainsi au départ beaucoup d'autres classes sont labellisées incorrectement comme des voitures, et à la fin des passes, les tracteurs et autres véhicules sont bien traités comme des fonds, mais une grande proportion des pick-up et des vans restent toujours labellisée comme des voitures même à la fin de l'apprentissage. Ce n'est pas surprenant, compte tenu de la grande proximité visuelle entre pick-up, voitures et vans.

2.2.3 Limitations de notre détecteur

Notre détecteur possède certaines limitations. La première est la faible résolution spatiale de la carte obtenue du réseau, qui impose que le centre de la détection trouvée

se trouve sur une grille. Compte tenu des vérités terrains, il se peut qu'à cause de la quantification spatiale, nous manquions une ou plusieurs cibles. Ceci a été étudié sur une base interne. La Figure 2.11 montre la grille correspondant à un pas de 4 pixels marqué par des croix noires, ainsi que la carte dense obtenue en réduisant le pas de la fenêtre de détection. La tâche n'est pas toujours bien centrée. Bien sûr, il est possible de réaliser des stratégies de pas adaptatifs avec un pas plus fin dans les zones chaudes, mais cela ajoute des étapes supplémentaires. De plus, les fenêtres de détections obtenues sont de taille fixe. La Figure 2.12 montre le résultat de la chaîne actuelle. C'est pourquoi la plupart des détecteurs "single-shot" modernes utilisent ce qui s'appelle *la régression de boîtes englobantes*. Il suffit de faire apprendre au réseau en plus du score de classe pour une vignette donnée à régresser un offset/un décalage par rapport à la vraie position. Dans ce cas, nous nous plaçons encore en apprentissage supervisé, mais nous passons dans le cadre de la régression. Le problème est beaucoup plus dur, car nous souhaitons non pas une probabilité, mais une valeur scalaire bien précise, ce qui pousse Gidaris [68, 69] à reparamétriser la régression de boîtes comme un problème de classification (plus d'informations sur le sujet peuvent être trouvées au chapitre suivant). Nous pourrions garder notre réseau complètement convolutionnel, mais il faudrait ici apprendre une carte des décalages pour chaque pixel de la carte de sortie. Nous irions donc prédire une probabilité (vecteur de taille 2) et un décalage en x et en y la carte de sortie serait donc de profondeur 4. En utilisant, là encore, complètement les possibilités du tout convolutionnel. Une autre limitation est le champ réceptif de taille fixe de notre détecteur : les véhicules sont toujours cherchés dans des fenêtres 44x44, mais ce n'est pas un problème majeur, car la distance sol-capteur étant connue, les véhicules n'ont pas une variabilité extrêmement forte en termes de taille. Nous verrons dans la suite comment l'état de l'art des détecteurs CNNs à un seul étage permet de répondre à ces manques.

2.3 Notre détecteur face aux meilleurs réseaux Single-Shot

Le but de cette section est de comparer les mécanismes utilisés dans notre détecteur à ceux qui sont intégrés dans les autres détecteurs à un étage de la littérature du domaine pour identifier les possibles différences, avantages et inconvénients de chacun.

2.3.1 YOLO (v1)

Overfeat [210] fut peut-être le premier de cette longue lignée de détecteurs *tout convolutionnels*, qui sont maintenant les solutions de départ dans toutes les compétitions de détection d'objets. Nous avons choisi pour commencer l'un des plus emblématiques détecteurs single-stage : YOLO (You Only Look Once) [192]. Il nous a séduits du point de vue de sa simplicité et du fait qu'il est une très bonne illustration de la conclusion de la section 2.2.1. YOLO, donc, structure **quasiment** complètement convolutionelle, va fournir pour une image donnée un nombre de boîtes englobantes fixé. Une image de taille fixée va fournir une carte convolutionelle de taille $S * S * D$. S représentant les dimensions spatiales et D la profondeur de la carte convolutionnelle. Cette carte se re-projette dans l'image de départ comme expliqué précédemment pour la découper en SxS zones, qui ne se recouvrent pas. A la différence de notre réseau, les informations de ces différentes zones sont alors mises en commun au travers d'une couche complètement connectée pour produire une sortie finale $S * S * N$ où N représente la dimension des prédictions finales de notre réseau que nous allons expliciter. Chaque cellule va prédire B boites, chacune décrite par 4 scalaires $(x, y, \sqrt{w}, \sqrt{h})$ ou x et y représentent la position du centre *relativement à la cellule*, w et h respectivement la largeur et la hauteur de la boîte normalisée par la taille de l'image. L'ajout de la racine vient du fait que le gradient obtenu en utilisant un coût L2 classique de régression scalaire directement sur w et h donnerait des gradients trop forts par rapport aux autres si la boîte est plus grande (comme évoqué dans la section critère détection). Cela est légèrement diminué en régressant les racines, car la fonction racine croît lentement et donc les défauts de localisation de deux boîtes de tailles différentes sont rapprochés artificiellement, mais cela est légèrement insatisfaisant.

Pour chacune de ces B boîtes définies par la paramétrisation ci-dessous, l'auteur prédit un score de présence/absence d'objets. De plus, pour chaque cellule de la grille, il prédit C scores de classes *indifféremment des boîtes*. Cela nous fait donc en profondeur $N = (B * 5 + C)$ cartes disposées dans l'ordre souhaité.

La principale difficulté consiste à implémenter une logique dans l'entraînement de mise en correspondance des boîtes à régresser et à ranger, pour éviter les trop forts gradients. Les vraies différences par rapport à notre détecteur naïf sont :

— La régression de décalages,

— L'utilisation de coût L2 pour la classification (choix a priori non évident),

— La factorisation des scores en une partie objet et une partie classe,

— La mise en commun des prédictions aux différents endroits de la carte,

— Le pré-entrainement ImageNet [44].

Là encore, il convient de s'interroger sur la pertinence de chacun des choix de l'auteur. Comme cette chaîne de détection est assez simple il est assez facile de créer sa version de YOLO personnelle en utilisant par exemple une sigmoïde pour forcer les sorties de classification à être entre 0 et 1, etc. Nous pourrions aussi envisager d'implémenter les couches complètement connectées comme des convolutions et d'avoir pour des images plus grandes une sortie comportant des grilles 7x7 entrelacées, ou même de changer l'architecture pour avoir une grille plus fine. Les faibles performances de YOLO dans sa première version comparées à Faster R-CNN [196] par exemple ont d'ailleurs été corrigées par l'auteur en incluant des séries de modifications successives de paramétrisation et d'architectures non loin de celles que nous avions suggérées [193, 194].

Il y a cependant une composante de YOLOv1, que partagent tous les systèmes de détection état de l'art et qui n'est pas encore contournable facilement : le pré-entrainement ImageNet. Entraîner un détecteur from scratch est possible, mais les performances obtenues après un pré-entrainement sont quasi systématiquement meilleures [71, 41, 144, 87, 88, 150, 192].

Deux travaux récents cependant sortent du lot et montrent qu'en choisissant savamment l'architecture du réseau et en surveillant de près le procédé d'apprentissage, il est possible dans certains cas de rattraper le pré-entrainement comme dans [212, 213].

2.3.2 SSD

L'un des inconvénients de YOLO, qu'il partage avec notre détecteur, vient des couches de pooling. La résolution de la sortie est faible. Les petits objets sont donc difficilement localisables : l'information précise de localisation présente dans les premiers niveaux des couches convolutionnelles est floutée au fur et à mesure par le pooling. SSD [150] part de ce principe en utilisant une idée, que l'on peut trouver aussi chez Shelhamer [211] déjà cité, et qui ne cesse pas d'être réutilisée en segmentation sémantique et détection d'objets [197, 84, 132, 144] : celle de *réutiliser l'information de localisation des premières couches pour la fusionner avec celle des couches supérieures au contenu plus sémantique et discriminant*. SSD est non seulement plus rapide que le YOLO de base (car il utilise des images moins résolues en entrées 300x300 dans sa version rapide vs 448x448 pour YOLO), mais aussi plus précis. SSD introduit aussi un concept, qui sera étudié plus en détail dans les chapitres suivants : les ancres. Dans YOLO les boîtes englobantes sont prédites directement en coordonnées grille, ce qui rend l'entraînement très instable (YOLOv1 ne converge sur VeDAI qu'en opérant un découpage préalable des images). En s'inspirant des réseaux à résidus (voir Annexe A au besoin), nous pouvons apprendre à régresser le résidu/la différence entre un a priori fixe (que nous appellerons une ancre) et la vraie boîte. Ainsi, l'optimisation sera facilitée. C'est ce que fait SSD.

Pour chaque boîte, nous prédisons à partir des "features" situées au même endroit une déformation, *la déformation apprise dépend donc de la forme de l'a priori*. L'a priori peut être carré de la taille exacte du champ réceptif de la carte considérée ou complètement différent par exemple rectangulaire.

De plus, SSD réalise la prédiction de boîtes englobantes *à différentes échelles en utilisant des cartes convolutionnelles à différents niveaux de résolution spatiale*. Ainsi SSD utilise des ancres assez grosses sur les derniers niveaux des features maps très peu fins, et des plus petites sur les plus proches de l'image. Chaque carte qui est utilisée pour prédire des ancres devient donc spécialisée en une taille d'objets. Cela suit quand même de loin l'évolution de la taille des champs réceptifs. Là aussi, l'assignation des ancres à leurs objectifs pendant l'entraînement est cruciale. Comme pour Wide-LeNet-5 il faut décider quelles ancres utiliser pour les négatifs et positifs pendant l'apprentissage parmi

méthode	cars	camping cars	pickups	tractors	trucks	vans	boats	others	planes	moyenne
SSD1024	34.23 ± 9.67	20.35 ± 12.8	30.77 ± 14.15	9.42 ± 8.97	13.01 ± 7.58	6.36 ± 8.18	3.68 ± 4.38	1.26 ± 3.14	1.85 ± 4.27	13.44 ± 8.88

TABLE 2.7 – SSD1024 mAP sur VeDAI (10 folds)

toutes celles disponibles.

Il est intéressant de réaliser que YOLO est un cas particulier de SSD où nous ne garderions qu'un seul niveau de prédictions et où nous rajouterions une couche complètement connectée. Cette observation peut être étendue à la très grande majorité des détecteurs CNNs qui ne sont en fait pour la plupart que des variations les uns des autres. C'est le cas au moins en tout cas de OverFeat [210], Faster R-CNN [196], SSD [150], YOLO [192] et R-FCN [41]. Il peut cependant y avoir un monde en termes de performance entre une architecture et une autre. C'est ce que nous verrons dans la suite.

2.3.3 Expérimentations de YOLO et SSD sur la base VeDAI

Pour SSD nous utilisons un modèle pré-entrainé sur MS-COCO disponible dans l'API de détection d'objets de Tensorflow que nous entraînons sur la base VeDAI. Le réseau utilisé est MobileNetv2 [204]. Nous entraînons SSD1024. Nous obtenons ainsi notre premier résultat multi-classes. La performance est représentée par rapport à la métrique VeDAI dans la table 2.7 : Nous obtenons donc nos premiers résultats multi-classes et avec des boîtes de tailles variables, quelques exemples sont présentés Figure 2.13. La mAP sur les voitures est comparable à celle obtenue par Wide-LeNet-5 sans stratégie de hard-mining (34 contre 28-30). Les mAPs sur les classes proches de voitures comme pick-ups et camping-cars sont du même ordre de grandeur. Les mAPs sur les autres classes ne sont pas bonnes probablement du fait du faible nombre d'exemples de ces classes dans la base d'apprentissage. Quant à YOLOv1 nous avons utilisé deux implémentations différentes celles de DarkNet [195] et celle de Darkflow [232] et ne sommes pas arrivés à le faire converger sur les images entières (une convergence à une mAP de l'ordre de 30% est observée en découpant les images en vignettes plus petites et en sur-échantillonnant). Nous avons été jusqu'à remplacer les fonctions de régression par des fonctions de Huber

mais l'instabilité liée à la fonction de coût est encore trop grande car YOLOv1 a été réglé pour des bases comme MS-COCO [143]. Cette section permet de conclure sur les performances atteignables sur notre base par les détecteurs à un étage et leurs limitations sur les petites bases de données comportant des classes difficiles à séparer.

2.4 Conclusions et perspectives

Ce chapitre nous a permis de proposer et de valider expérimentalement un détecteur à un étage qui donne des performances intéressantes sur la base VeDAI ; ces performances sont nettement au-delà ce qui avait été obtenu sur cette base dans le passé. Nous avons aussi identifié les composantes manquantes à notre détecteur par rapport à ses concurrents à un étage : la régression de décalage permettant de régler la taille des boites (présente dans YOLOv1 et SSD), l'utilisation de cartes convolutionnelles de plusieurs niveaux de résolution (dans SSD), ainsi que l'entraînement sur images entières (SSD et YOLOv1). Nous avons également mis en avant les limites des détecteurs à un étage, la principale étant la difficulté à effectuer simultanément la classification objets et fonds ainsi que la discrimination entre les classes proches. C'est pour cela , par exemple, que YOLOv1 tente de différencier le score d'objet du score de classe. Plus généralement, les cascades à deux étages présentent un intérêt dans notre cas, en particulier pour les raisons suivantes :

— il est bien plus facile de discriminer tous les véhicules du fond, qu'une classe particulière de véhicule contre toute le reste (fonds+autres véhicules).

— Une fois que les véhicules sont normalisés en rotation, la classification est considérablement simplifiée (voir section 2.2.2).

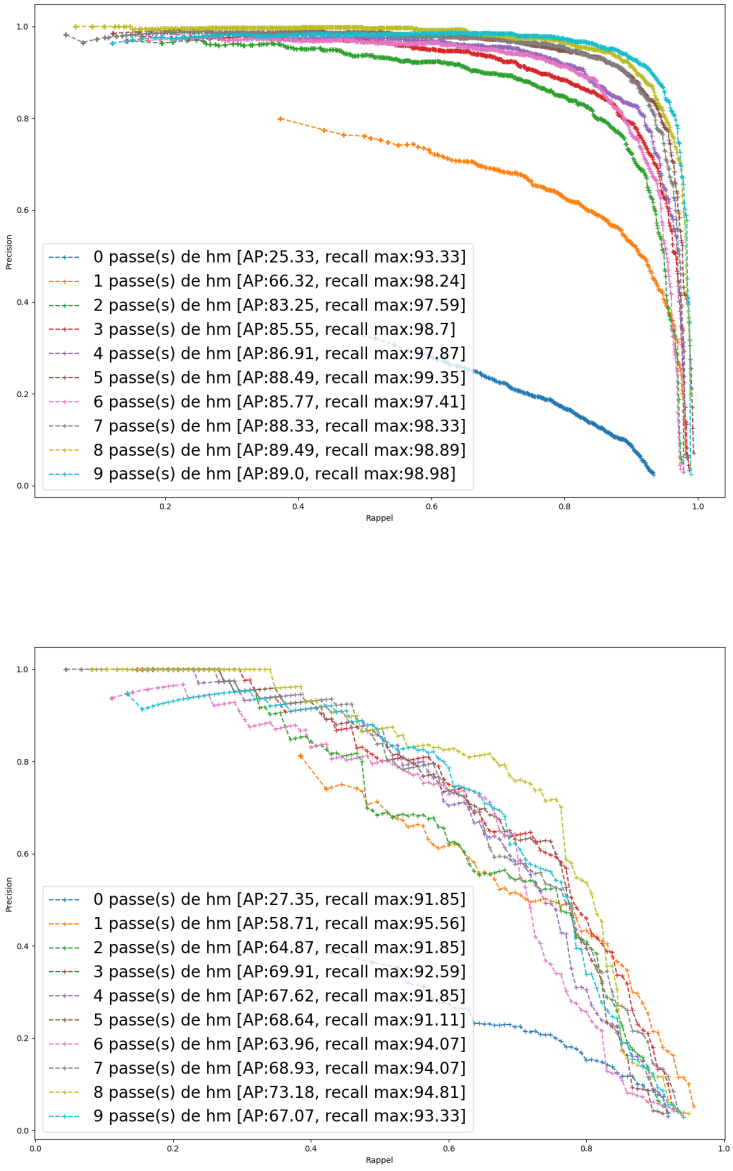


FIGURE 2.5 – Évolution des performances en détection sur l'ensemble d'apprentissage en haut et sur l'ensemble de validation en bas

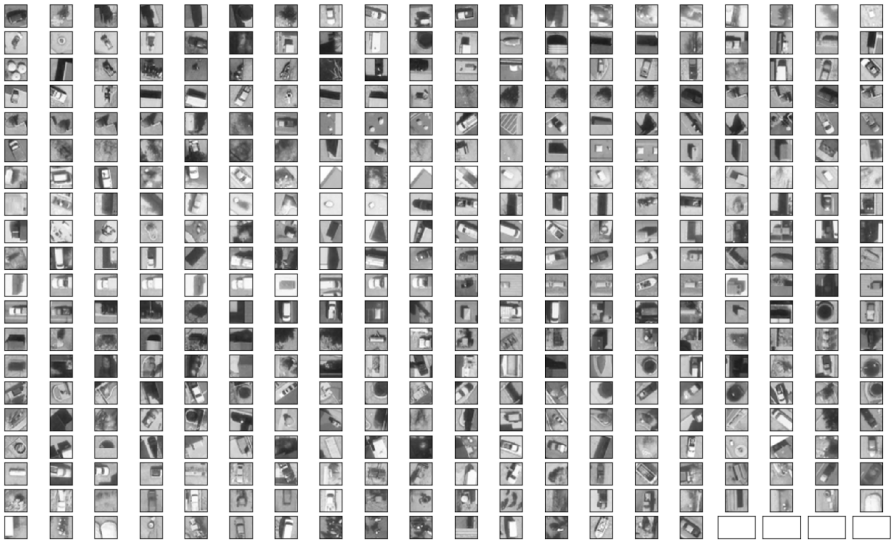


FIGURE 2.6 – Nous pouvons apercevoir dans cet échantillon d'autres véhicules, des bords de bâtiments, des trampolines, des piscines et d'autres fonds non triviaux

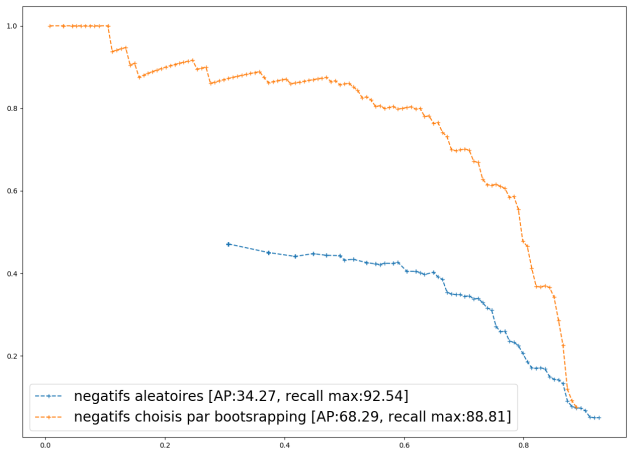


FIGURE 2.7 – Différences entre les performances obtenues à nombre d'exemples fixé, en choisissant les négatifs par bootstrapping ou bien en prenant des fonds au hasard

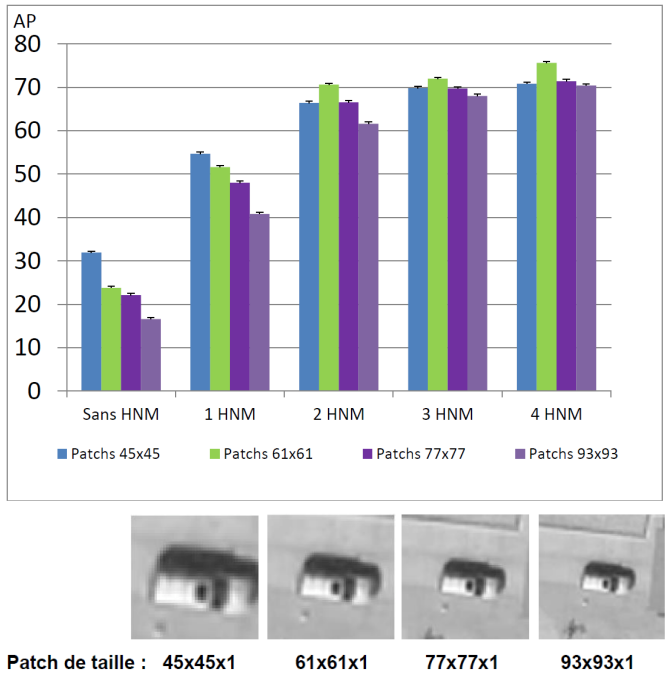
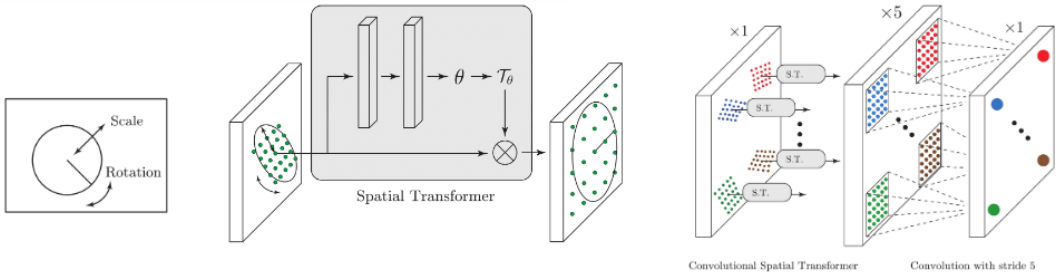


FIGURE 2.8 – Influence du contexte dans les performances en détection en fonction des passes de hard negative mining. En bas nous montrons ce que cela représente pour un patch de voiture



(a) SIFT (b) Spatial transformer (c) Convolutional spatial transformer

FIGURE 2.9 – Figure représentant le spatial transformer complètement convolutionnel tiré de [31]

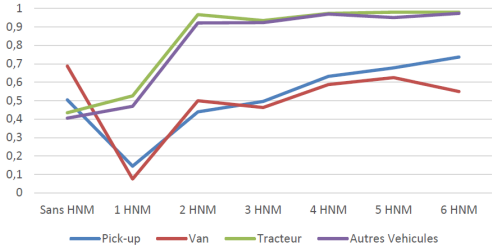


FIGURE 2.10 – Évolution des proportions des instances des différentes autres classes correctement labellisées comme fonds par notre détecteur de voiture

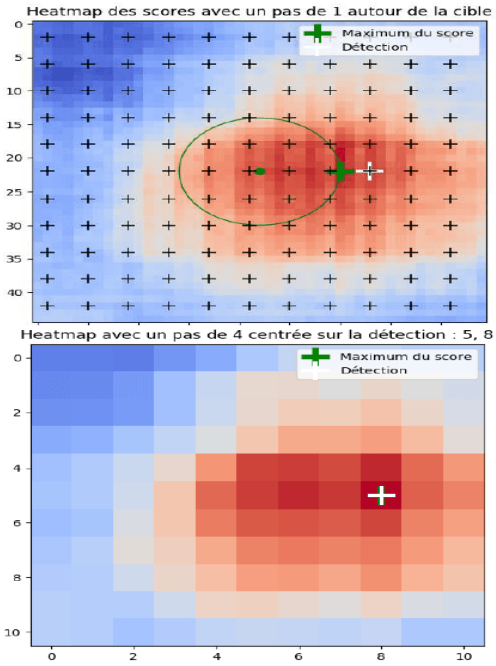


FIGURE 2.11 – Deux cartes de scores sont présentées. Celle du haut montre la carte précise obtenue avec un pas de 1 dont le maximum (croix verte) est dans la cible, le détecteur étant effectivement appliqué avec un pas de 4 trouve un autre maximum proche (croix blanche) et manque la cible. La carte en bas, effectivement obtenue par le détecteur, explique pourquoi.

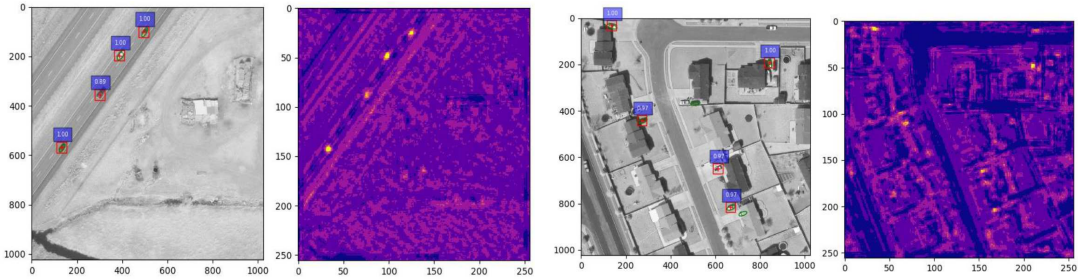


FIGURE 2.12 – Nous présentons deux exemples de détections de Wide-LeNet-5 après NMS ainsi que les cartes de scores correspondantes

FIGURE 2.13 – Nous présentons deux exemples de détections de SSD1024 après NMS

Chapitre 3

Premier détecteur en cascade utilisant un mécanisme d'inférence de l'orientation des véhicules

Nous avons présenté dans le chapitre précédent une méthode à un étage et conclu, sur la base de nos expérimentations, que les méthodes à deux (ou plus) étages devraient permettre d'améliorer les performances de notre détecteur. C'est l'objet de ce chapitre.

Ce chapitre présente un détecteur à deux étages basé sur l'alignement en rotation des vignettes. En effet, nous avons vu au chapitre précédent que la classification de véhicules est considérablement simplifiée lorsqu'il est possible d'aligner tous les véhicules sur une même orientation. Nous proposons donc une méthode d'inférence de l'angle de rotation des véhicules, basée sur un principe de régression d'angle. Cette méthode permet d'aligner l'orientation des véhicules puis de les re-classer, ce qui est désormais plus facile. Ce chaînage de plusieurs classifieurs, appelé cascade, est utilisé dans la plupart des algorithmes de détection modernes.

3.1 Objectifs

Dans ce chapitre nous réalisons la cascade à deux étages décrite dans l'introduction du chapitre. Nous avons tous les blocs en main pour réaliser cette cascade, sauf le bloc d'estimation de l'angle de rotation d'une vignette permettant de normaliser l'orientation des véhicules. Prédire l'orientation d'une vignette est lié à un problème de régression. Nous allons utiliser les mêmes structures CNNs pour résoudre ce problème, mais cette fois-ci, dans une optique de régression d'une grandeur scalaire. Après un court état de l'art sur la régression en vision par ordinateur, nous utiliserons une approche naïve pour régresser l'angle et en tirerons des conclusions. Motivés par cette connaissance, nouvellement acquise, nous utiliserons de la régression de Distribution à Distribution (D2D) [171] pour construire le bloc final, en montrant expérimentalement son avantage sur la régression classique Point à Point (P2P). Enfin nous utiliserons ce bloc dans la cascade que nous testerons sur la base VeDAI.

3.2 Régression et vision par ordinateur

Il existe beaucoup de travaux sur la régression en vision par ordinateur. Dans les années 90, elle était surtout utilisée pour l'estimation de pose 3D comme dans [83]. C'est aujourd'hui un composant essentiel de toutes les chaînes de détection, en particulier au sein de l'étape de régression de boîtes englobantes. Elle se trouve aussi dans l'estimation de position de caractéristiques de visages (coins des yeux, bouches, etc.) dans [248], l'estimation de pose de visage [11], celle de modèles 3D humains [18], dans l'estimation d'âge [57] et encore bien d'autres. En plus de [149], on trouve quelques travaux appliqués à l'estimation d'orientation de petits véhicules par exemple dans [95] où les auteurs utilisent des convolutions déformables (en déformant l'image puis en appliquant une convolution normale pour plus d'efficacité). Dans [74] des réseaux équivariants à la rotation sont utilisés. Compte tenu des difficultés d'optimisation de réseaux profonds en régression relevés par Karpathy, dans le domaine de l'estimation d'orientation par [165], qui propose de régresser plutôt les cosinus et sinus plutôt que directement l'angle et enfin par [81], qui prône le passage en classification aux dépens de n'importe quelle approche

filter	type	input size	output size
16 5x5x1	convolution	45x45x1	45x45x16
45x45	Avg-pooling	45x45x16	16
16x4096	fully-connected	16	4096
4096x4	fully-connected	4096	4
None	Softmax	4	4

TABLE 3.1 – Corps du premier réseau de classification d'angles.

basée sur de la régression directe ou indirecte, nous choisissons donc d'entraîner des réseaux en classification.

3.3 Premiers réseaux de régression

Nous proposons ici une première méthode simple où la régression est approximée par une classification. La regression est donc quantifiée. Nous prenons pour cela une architecture de classification avec 4 classes ou bins. [78] est le premier, à notre connaissance, à considérer cette re-paramétrisation.

— 0 à 36 ou 144 à 180 (classe 1),

— 36 à 72 (classe 2),

— 72 à 108 (classe 3)

— 108 à 144 (classe 4).

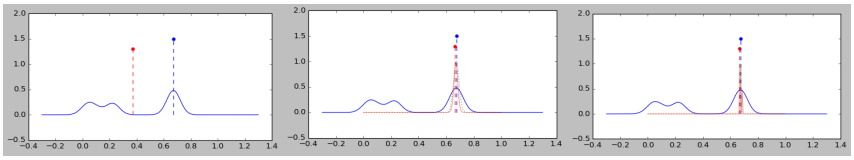
La première classe permet de résoudre la π-périodicité. Notre architecture fait intervenir 16 filtres de convolutions afin d'implémenter des gradients par rapport aux 4 orientations, suivis d'un global average pooling et de couches complètement connectées.

Nous entraînons ce classifieur sur 16000 imagettes de voitures et testons sur environ 4000 voitures. Nous utilisons de l'augmentation de données en faisant tourner chaque voiture présente dans le batch d'un angle uniformément distribué dans $[0, \pi]$ (ce qui change le label bien sûr). Le corps de notre premier réseau est contenu dans la Table 3.1.

Notre réseau ne peut pas sur-apprendre l'ensemble d'apprentissage ceci est visible sur la table 3.1(même en augmentant le nombre de filtres dans la première couche). Nous identifions les cas d'échecs comme étant inhérents à la structure de notre réseau. Les

Coût/Précision	baseline GAP	baseline AP	baseline MP local
Training	0.3351/86.52	0.119/95.68	0.1207/95.39
Validation	0.3497/86.72	0.1594/95.15	0.1828/94.31

TABLE 3.2 – Effets des différentes variantes architecturales sur les performances en classification d'angles. GAP : Global Average Pooling, AP : Average Pooling et MP : Max-pooling



(a) (b) (c)

FIGURE 3.1 – Visualisation d'un exemple, issu de nos données jouet, pour 3 méthodes différentes évaluées : (a) P2P, (b) D2D avec modèle gaussien, (c) D2D avec modèle de Dirac. La courbe bleue est l'observation, la rouge, la distribution prédite tandis que les points bleus et rouges sont respectivement la vérité terrain et les valeurs prédites des paramètres devant être estimés.

voitures possèdent, malheureusement pour nous, aussi des gradients orthogonaux à ceux donnant l'orientation. Ceci s'explique par la présence de vitres et de toits. En faisant la moyenne sur toute l'image, il est par conséquent difficile de discriminer quels gradients viennent de l'orientation principale (celle qui est recherchée). Il est donc mal venu de se débarrasser totalement de l'information spatiale dans la réponse des filtres. Nous remplaçons donc le global average-pooling GAP par un average pooling AP local par cellule 2x2. Les résultats sont bien meilleurs. Les résultats sont récapitulés dans la table 3.2.

Cette observation sur la multi-modalité des gradients des vignettes de voitures nous oriente vers la régression de Distribution à Distribution (D2D) [171].

3.4 Aller plus loin avec la régression : la régression D2D

3.4.1 Analyse de la régression standard

La plupart des régressions standards dites de point à point (P2P) s'appuient sur la minimisation d'un problème des moindres carrés. En appelant N le nombre d'exemples

d'apprentissage, donnés sous la forme de paires (x_i, y_i) avec x_i l'observation et $y_i \in \mathbb{R}$ les paramètres à inférer, et W les paramètres du modèle (c.-à-d. ceux du régresseur), le régresseur $f(x; W)$ est obtenu en minimisant la fonction de coût $J(W)$ suivante :

$$J(W) = \frac{1}{N} \sum_{i=1}^{N} \|f(x_i; W) - y_i\|. \tag{3.1}$$

Le coût moindres carrés (LSE) est obtenu quand $\|.\|$ est le carré de la norme L2. Quand $\|.\|$ est la norme L1, on l'appelle le coût Moindre Déviation Absolue (LAD).

Nous illustrons la méthodologie précédente avec un problème jouet pour lequel nous avons construit une BDD de un million d'exemples (2/3 pour l'apprentissage, 1/3 pour le test). L'observation x est construite à partir d'un mélange de 4 gaussienne unidimensionnelles, quantifiées en 256 bins. La courbe bleue Fig. 3.1 correspond à un seul exemple de cette base. Le paramètre à régresser (y) est la moyenne de la gaussienne avec le plus de poids dans le mélange. Sur la Fig. 3.1, cette valeur est marquée par un point bleu. Le point rouge indique la valeur prédite par le modèle. En pratique, sur cette BDD jouet, nous utilisons comme fonction f un simple réseau de neurones avec 2 couches et une couche cachée avec 10 neurones suivi d'une activation non linéaire (ELU [33]). Pour la régression point à point (P2P) la sortie est de taille 1, la même que celle de la valeur à prédire.

La Fig. 3.1-(a) illustre un des problèmes majeurs *de la régression point à point* avec des coûts LSE ou LAD. Quand l'observation possède des ambiguïtés –cela est représenté dans le problème jouet par l'addition de 3 autres gaussiennes– ou quand elle a plusieurs modes, le régresseur tend à prédire une moyenne des différents modes. La plupart du temps, la valeur prédite (les points rouges sur la Fig. 3.1) se situe entre les modes au lieu d'être sur le mode principal.

Nous donnons dans la Table 3.3 une analyse quantitative des performances de la régression point à point sur cette base.

3.4.2 De la régression point à point (P2P) à la régression de distribution à distribution (D2D)

Nous expliquons maintenant comment passer de la régression point à point à de la régression entre distributions. La sortie du régresseur est maintenant une distribution, c'est à dire un vecteur 1D de valeurs réelles positives sommant à 1. Des formulations plus générales sont possibles, mais ne sont pas l'objectif de cette thèse. Une telle reparamétrisation pose deux questions. La première est : comment produire la distribution de l'ensemble d'apprentissage ? La deuxième : comment prédire le paramètre optimal à partir de la distribution prédite ?

Vis-à-vis du premier point, nous introduisons une fonction $g(z; y_i)$ introduisant un mapping entre un point et une distribution (discrète). Nous proposons deux fonctions candidates différentes. La première est simplement comme précédemment un dirac centré en y_i, écrit comme :

$$g(z; y_i) = \mathbb{1}_{[z=y_i]} \tag{3.2}$$

Une alternative est une distribution gaussienne centrée en y_i :

$$g(z; y_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{y_i - z}{\sigma}\right)^2} \tag{3.3}$$

Où σ est un paramètre cross-validé et fixé une fois pour toutes (Il est intéressant de remarquer que nous pourrions tout aussi bien prédire σ à l'inférence).

Pour chaque vérité terrain y_i nous définissons la fonction de distribution $g : g(z, y_i) \propto [g(z_1; y_i), ..., g(z_k; y_i), ..., g(z_D; y_i)]$, où $g(z_k; y_i)$ est la version discrète de $g(z; y_i)$ (avec $\sum_k g(z_k; y_i) = 1$), $g(z_k; y_i)$ étant la probabilité selon nos hypothèses que y_i prenne la valeur z_k. Remarquons qu'en choisissant un modèle paramétrique, nous nous retrouvons avec deux hyper paramètres D et σ en plus.

Dans cette section, nous considérons uniquement Eq. (3.3), le cas Eq. (3.2) est traité dans la section suivante (car il est problématique). Cependant, si cela fait sens, cela pourrait théoriquement être étendu à n'importe quelle distribution, ce qui est intéressant si l'utilisateur a un a priori fort sur la distribution des y.

La deuxième question est d'inférer la valeur optimale à partir de la distribution prédite. Un moyen très simple de le faire est de prendre la valeur pour laquelle la distribution est maximale. Nous appelons $h(x_i; W) \in \mathbb{R}^D$ la sortie du régresseur et W ses paramètres , et $h_k(x_i; W)$ sa k-ème composante.

$$\hat{y_i} = \frac{1}{D} \arg\max_k h_k(x_i; W) \tag{3.4}$$

Une autre option est de prendre l'espérance :

$$\hat{y_i} = \frac{1}{D} \sum_{k=1}^{D} k \times h_k(x_i; W) \tag{3.5}$$

Nous n'observons pas de grande différence entre les deux sur notre base de données jouet, comme nous pouvons voir Fig.3.3, mais l'inférence argmax donne de légèrement meilleurs résultats.

Les paramètres W du régresseur $h(x; W)$ sont obtenus en minimisant la distance KL entre la sortie et les distributions vérité-terrain que nous avons créées.

$$J(W) = \frac{1}{N} \sum_i KL(h(x_i; W), g(z; y_i)) = \frac{1}{N} \sum_i \sum_j h_j(x_i; W) \ln \left(\frac{h_j(x_i; W)}{g(z_j; y_i)} \right) \tag{3.6}$$

Dans le cas particulier où le régresseur $h(x_i; W)$ est un modèle gaussien de moyenne W_m et d'écart-type W_σ, la distance KL peut s'écrire [96] :

$$KL(h(x_i; W), g(z; y_i)) = \frac{1}{2} \left(\ln \frac{W_\sigma}{\sigma} + \frac{\sigma}{W_\sigma} + \frac{\|W_m - y_i\|^2}{\sigma} + D \right) \tag{3.7}$$

Dans ce cas particulier, les optimisations peuvent se faire de manière indépendante pour W_m et W_σ. Les valeurs optimales sont obtenues en minimisant un problème aux moindres carrés, de la même manière que celui de l'eq 3.1 présenté dans la section précédente. LSE peut être vu comme un cas particulier de D2D, si les deux distributions sont modélisées par une seule gaussienne.

En ce qui concerne les expériences sur notre base jouet, nous pouvons utiliser le même réseau de neurones qu'à la section précédente sauf que la sortie a maintenant

	régression P2P	régression D2D(gaussienne)	régression D2D (Dirac)
Inférence directe	12.54 ± 0.15	—	—
Espérance – Eq. (3.5)	—	2.26 ± 0.008	2.86 ± 0.24
inférence argmax – Eq. (3.4)	—	2.04 ± 0.01	2.80 ± 0.24

TABLE 3.3 – Mean Average Error (MAE) sur notre base jouet avec 3 différentes méthodes : (a) régression point à point (P2P), (b) de distribution à distribution (D2D) avec un modèle gaussien et une distance KL, (c) de distribution à distribution (D2D) avec un modèle de Dirac et l'entropie croisée. Les nombres ont été multipliés par un facteur 10^2 par rapport à la figure.

D composantes. Dans ces expériences D vaut 128 et σ vaut 0.02 (les valeurs étant choisies par validation-croisée), et le coût est donné par l'Eq. (3.6). L'inférence est faite en utilisant ou l'argmax, ou la moyenne. La performance est bien meilleure qu'avec la régression points à points, comme on peut le voir Table 3.3. La différence est aussi visible sur la Fig. 3.1 ou la distribution prédite est très proche de celle du mode principal. Nous remarquons aussi que la valeur prédite est très proche de la valeur cible.

3.4.3 Régression D2D avec un a priori de Dirac

Quand les annotations de points de la BDD jouet sont converties en distribution en utilisant l' Eq. (3.2), la régression de distribution à distribution peut être vue comme un problème de classification comme à la section précédente. Les valeurs continues y_i sont en effet représentées par D labels de classes mutuellement exclusifs. Dans ce cas il est possible d'utiliser un coût de classification comme l'entropie-croisée :

$$J(W) = -\frac{1}{N} \sum_i \sum_j g(z_j; y_i) \ln h_j(x_i; W) \tag{3.8}$$

Il faut noter la relation entre Eq. (3.6) et Eq. (3.8). En termes de résultats sur notre data-set jouet, la performance est légèrement moins bonne qu'avec la distribution gaussienne, mais bien meilleure qu'avec la régression point à point comme le montre la Fig. 3.1 et la Table 3.3. La variance obtenue avec la distribution gaussienne est bien plus petite que celle obtenue avec le Dirac et elle est moins sensible au choix de D, ce qui constitue deux gros avantages.

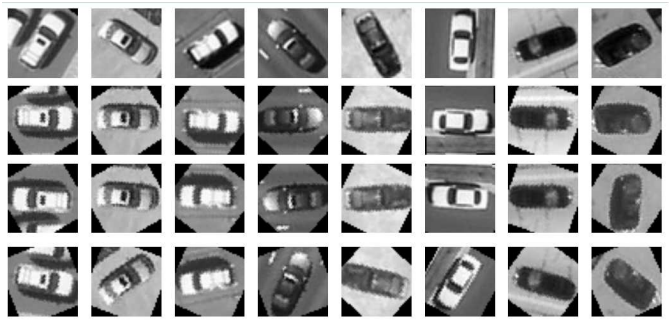


FIGURE 3.2 – Alignement de l'orientation des véhicules de VeDAI : véhicules tels quels (première ligne), véhicules alignés avec l'angle vérité terrain (2ème ligne), alignement avec les régressions D2D-Gaussienne (3ème ligne) et P2P (4ème ligne) .

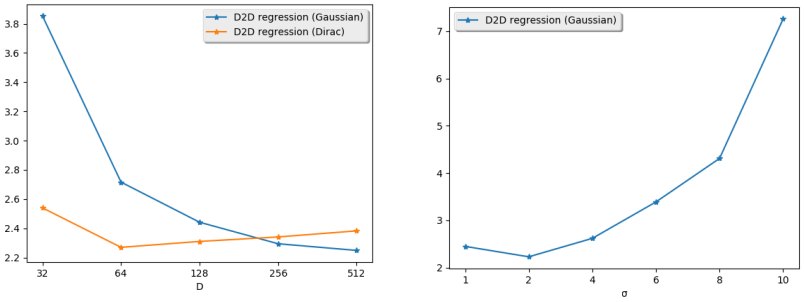


FIGURE 3.3 – A gauche, nous présentons la performance de la régression D2D avec la distribution gaussienne d'écart type fixé $\sigma = 2$ (bleu) et avec la distribution de Dirac (orange), en fonction de D. A droite est présentée la performance de la régression D2D (distribution gaussienne) avec $D = 512$ en fonction de σ.

3.5 Comparaison D2D et régression points à points sur notre problème d'orientation de véhicules

Les expériences de cette section valident la méthodologie précédente en comparant la régression points à points P2P avec la régression D2D proposée. Nous étudions donc l'estimation de l'orientation d'un véhicule dans une région d'intérêt trouvée par notre détecteur, comme illustré Fig. 3.2.

Nous utilisons la même base de données d'orientations de voitures que précédemment, mais la métrique est maintenant la Mean Average Error (ce n'est plus une métrique de classification).

Les différents gradients des voitures étant analogues aux différents composants du mélange de gaussiennes de notre problème jouet, nous espérons observer les mêmes résultats.

Regression	taille de la distribution (D)	Coût	MAE
P2P	—	L2	23.91 ± 2.42
P2P	—	L1	23.76 ± 2.48
D2D (Dirac)	64	CE	2.27 ± 0.02
D2D (Gaussian)	512	KL	2.23 ± 0.05

TABLE 3.4 – Estimation de l'orientation du véhicule : Mean Average Error (en degrés) pour les différents types de régresseurs considérés.

Nous reportons dans cette section les performances des 4 régresseurs présentés précédemment. La Table 3.4 donne les performances des régresseurs en termes de MAE (en degrés avec écart-type) sur la tâche d'estimation d'orientation. Nous avons essayé beaucoup d'architectures, la meilleure étant un CNN 5 couches avec des activations ELU [33]. Le réseau choisi possède 3 couches conv+max-pool suivies par une couche cachée. Pour chaque formulation (P2P L1, etc.) nous avons sélectionné les meilleurs hyper-paramètres (taux d'apprentissage, régularisation L2, dropout, etc.) sur un ensemble de validation, et nous les utilisons pour reporter la moyenne de 5 entraînements et l'écart-type associé. Là encore, les régresseurs ayant les meilleures performances sont les régresseurs D2D avec un léger avantage pour celui avec distribution gaussienne. Ceci étant probablement dû à la multi-modalité des images. La régression P2P essaye de remplir des objectifs contradictoires et donc donne la moyenne des résultats, ce qui n'est pas le mode recherché. En utilisant une distribution nous sommes capables d'abaisser la MAE à trois degrés, ce qui est proche d'une précision pixellique. Nous regardons aussi les influences comparées de nos deux hyper paramètres D et σ sur les réseaux D2D Figure 3.3. Même quand il y a un faible nombre de classes la performance des réseaux D2D est bien meilleure que celle des réseaux en régression point à point : avec seulement 8 classes la MAE est de 6.71 ± 0.95 degrés, ce qui est bien meilleur que la régression P2P. Comme sur la base jouet, l'utilisation de la distribution gaussienne est bien meilleure que celle avec la distribution Dirac ceci est visible table 3.4.

Regression	Côut	AP	RAPPEL@0.01FPPI
—	—	73.04	26.15
P2P	L1	73.35	32.69
D2D (Gaussien)	KL	77.49	44.99
D2D (Dirac)	CE	79.14	48.62
Oracle	None	89.14	50.32

TABLE 3.5 – Détecteur cascade sur la classe voiture

3.6 Utilisation de la brique d'estimation d'angle dans notre première cascade

Maintenant que nous avons un moyen d'estimer l'angle, ainsi qu'un réseau appris sur des cibles rectifiées. Nous utilisons le détecteur décrit dans le Chapitre 2 avec un seuil de détection très faible afin d'avoir un excellent rappel. Ensuite, nous extrayons les vignettes de détection puis nous les faisons passer dans le second réseau d'estimation d'angle (entrainé sur les vignettes de véhicules disponibles), nous utilisons la valeur régressée pour rectifier tous les patchs (y compris les fonds). Ensuite notre deuxième réseau final (qui est une copie du premier réseau de classification) reclasse les détections et nous reportons le résultat de la chaîne de détection finale. Nous incluons dans le tableau la borne supérieure de la performance de la chaîne avec un oracle connaissant les vrais angles de chaque cible et un régresseur aléatoire sur les fonds. La Table 3.5 montre que quelle que soit la régression employée le gain en termes de performance de détection est notable. La D2D en Dirac permet de gagner 10 points d'AP ce qui est très encourageant pour la suite.

3.7 Conclusions

Nous avons montré que l'estimation d'angle par deep-learning était possible, même si nous avons mis en exergue la difficulté de prédire des gradients dans des images multi-modales. Nous avons évalué différentes alternatives et montré la supériorité de l'approche de régression de distribution à distribution (D2D) pour gérer ce problème. De plus nous avons démontré l'avantage d'une cascade intégrant cette nouvelle régression d'angle dans la détection de petits véhicules montrant un gain de plus de 10 points en AP

sur VeDAI lorsque l'angle est parfaitement connu (89 contre 77 en moyenne). Cependant, nous n'avons toujours pas intégré les avancées présentes dans les détecteurs de l'état de l'art : il n'y a pas encore de régression de boîtes englobantes et l'entraînement s'effectue encore sur vignettes et par étape successives, ce qui rend les expériences et l'inférence très lourdes. Pour entrer dans le paradigme de l'entraînement complètement end-to-end (d'un bout à l'autre) nous utiliserons dans la suite une nouvelle architecture cascade état de l'art que nous améliorerons.

Chapitre 4

Second détecteur en cascade utilisant des ancres tournantes

Dans ce chapitre, nous cherchons à améliorer le détecteur Faster R-CNN [196], qui est désormais le détecteur de référence dans la littérature du domaine. Notre objectif est de le rendre invariant aux rotations des véhicules dans les images. Plutôt que d'inférer l'angle directement, comme nous l'avons fait dans le chapitre précédent, nous utilisons ici un principe d'ancres tournantes (les fenêtres des images ne sont plus alignées avec le repère de l'image). Nous avons conçu cette méthode dès le début avec l'optique de l'intégrer au détecteur Faster RCNN déjà existant.

Avant de présenter cette contribution, ce chapitre commence par rappeler ce qu'est le détecteur Faster R-CNN [196].

4.1 Introduction à Faster RCNN

Nous nous proposons ici d'introduire le système de détection Faster R-CNN [196] sur lequel une partie de nos travaux sera basée. Nous évoquons ici les différentes avancées techniques ayant mené à sa création.

4.1.1 R-CNN [71]

Ross Girshick, un des auteurs du DPM [56], construit le premier détecteur CNN à devenir un standard : R-CNN [71] (OverFeat étant devenu un standard en termes d'architecture pour la classification plus qu'un système de détection). Une des raisons de sa popularité vient de l'utilisation d'un framework de Deep Learning : Caffe [120], qui implémente le gradient des briques classiques des CNNs. Cela permet la modification et la réutilisation des briques développées. Le rôle de ces frameworks pour la popularisation et le développement des méthodes de détection d'objets n'est pas à négliger. Girshick combine une étape non deep de recherche sélective [235] en effectuant de la segmentation de régions basée sur des indices de couleur et de gradients puis du regroupement hiérarchique (clustering) de ces régions. Le tout afin de générer des boîtes ayant une forte probabilité de comprendre un objet, car contenant des amas de pixels du même groupe. Nous montrons le résultat de l'implémentation de dlib [129] sur une image de VeDAI pour deux valeurs du paramètre `min_size` représentant la taille d'une région élémentaire sur la Figure 4.1 (nous pouvons filtrer les trop grandes boîtes en nous basant sur les gabarits connus des véhicules).

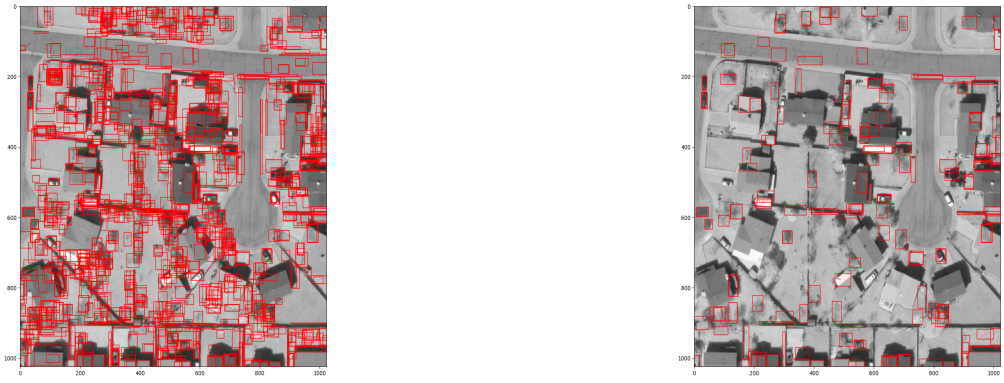


FIGURE 4.1 – Résultats de recherche sélective sur une image de VeDAI pour deux jeux de paramètres différents.

Ces régions candidates sont ensuite extraites avec p pixels de contexte puis warpées à la taille fixe IMAGENET 227*227. Ensuite, l'auteur remplace le softmax à 1000 classes

d'un réseau pré entraîné sur IMAGENET par le nombre de classes voulues et il fine-tune [1] le CNN en classification de régions. Le réseau n'est pas complètement convolutionnel ici, car l'étape de propositions de régions a déjà permis de réduire l'espace de recherche, et le parcours exhaustif ou quasi exhaustif n'est donc plus nécessaire.

Puis, sur chaque région, il apprend *sur les descripteurs gardés fixes* un SVM linéaire de classes ainsi qu'une régression linéaire de boîtes englobantes *par classe*. Les détections finales sont les régions initiales, déformées par la régression linéaire, de scores ceux du SVM multi-classes. Ainsi, même si le classifieur est très bon, il est crucial que l'étape de proposition de régions ait un bon rappel sinon des objets seront forcément manqués. Par contre, le fait que les régions aient peut-être un peu moins de 0.5 d'IoU avec les vérités-terrains est permis, car la régression linéaire peut apprendre un déplacement. Cette étape de proposition de boîtes génériques donne lieu à une littérature abondante. Nous citerons les 3 plus algorithmes les plus connus en dehors de la recherche sélective : BING [30, 264] reposant sur un modèle de gradient propre à toutes les régions contenant des objets, les boîtes de contours (EdgeBoxes) [269] basées sur des regroupements de contours/gradients et AttractioNet [69] utilisant des CNNs et une approche itérative (il en existe bien d'autres comme CSVM [263] ou OBN [4]).

La force des articles de Girshick réside dans la rigueur des analyses du design de son algorithme et par des études ablatives poussées. Par exemple, l'entraînement d'un SVM linéaire sur le descripteur pourrait paraître redondant car le CNN produit déjà des scores de classification. Seulement l'auteur montre que différents critères de définition d'un négatif et d'un positif doivent être utilisés en entraînant les couches basses du descripteur et en entraînant le SVM linéaire de classification pour aboutir aux meilleures performances. Une autre des forces de son algorithme est qu'une fois l'entraînement du CNN réalisé en classification, l'utilisateur peut pré-calculer les "features" de toutes les régions de l'ensemble d'apprentissage et du test. Encore une fois, une astuce de calcul se révèle décisive dans le design et le choix de l'algorithme. La dernière raison est que Girshick transforme, grâce à la recherche sélective, le problème de détection compliqué en un problème de classification pour lequel il peut s'appuyer sur les avancées des CNNs

1. ajuste finement ses poids

sans essayer comme OverFeat de résoudre tout d'un coup. Ce qu'il faut retenir de cette méthode est qu'il y a deux phases distinctes dans l'algorithme :

— la proposition de régions d'objets,

— la classification.

4.1.2 Fast R-CNN [70]

Après avoir construit R-CNN [71] et montré son avantage en termes de performances en détection par rapport au DPM [56], Girshick revient sur cet algorithme, qui nécessite plusieurs entraînements distincts ce qui est très long. Il utilise l'astuce du tout convolutionnel évoquée au chapitre précédent. Au lieu de warper les régions à la bonne taille puis d'extraire des descripteurs, il applique les premières couches du CNN sur l'image entière obtenant donc une carte convolutionnelle de résolution proportionnelle au nombre de couches de pooling. Puis, pour continuer le réseau en utilisant les propositions, Girshick invente un module permettant de faire du max-pooling dans la projection des régions obtenues par recherche sélective sur la carte convolutionnelle et d'obtenir, *quelle que soit la taille de la région sous-jacente* un descripteur de taille fixée comme avec le warping d'images de R-CNN, mais au niveau du descripteur cette fois-ci. **Ce module est capital pour la suite de la thèse. Il l'appelle RoI-pooling (Region of Interest Pooling)**. Il permet, *en gardant les propositions fixes*, de fine-tuner directement le CNN en classification et en régression de boîtes englobantes, en propageant le gradient dans l'image entière.

Sa définition mérite d'être faite rigoureusement : Une région étant définie par $(x_{min}, y_{min}, x_{max}, y_{max})$ en coordonnées entières, on divise la région dans le descripteur carte correspondant à cette région c.-à-d. $(\frac{x_{min}}{F}, \frac{y_{min}}{F}, \frac{x_{max}}{F}, \frac{y_{max}}{F})$ avec F le facteur de sous-échantillonnage du réseau, en $N_x * N_y$ blocs (7*7 dans l'article original). La taille des blocs (ou cellules) étant donc $\frac{x_{max}-x_{min}}{F*N_x}$ et $\frac{y_{max}-y_{min}}{F*N_y}$. Chaque cellule est ensuite quantifiée en prenant la partie entière inférieure du bord gauche et supérieure du bord droit. Il y a donc possible recouvrement des cellules. Le maximum est alors calculé sur tous les pixels en coordonnées entières inclus dans la cellule en question. La Figure 4.2 résume les opérations pour une cellule alignée avec les axes ABCD.

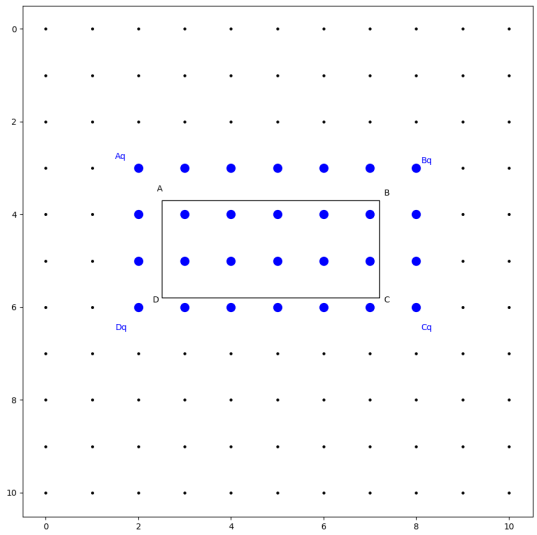


FIGURE 4.2 – RoI pooling classique sur une cellule : La grille des centres des pixels en noir, avec en petit les pixels non considérés, car n'étant pas dans la cellule. Les pixels finalement utilisés pour le maximum sont en bleu.

En plus de la valeur du maximum, l'indice dans la carte poolée du maximum est stocké pour permettre la back-propagation. Ces descripteurs, chacun de taille (N_y, N_x, D), sont passés ensuite dans des couches complètement connectées classiques pour donner un score de classe et régresser la position. L'entraînement se passe donc de la manière suivante :

— Pré-calcul d'un nombre fixé de régions par images d'entraînement par recherche sélective,

— Forward de l'image entière en utilisant le RoI pooling des régions pré-calculées sur la carte de convolution,

— Entraînement du réseau en classification et régression simultanément. Le réseau a deux branches (classification fine et regression), mais garde le même corps pour les deux.

Sans avoir besoin de briques supplémentaires on s'approche du end-2-end (entraînement de bout en bout) de YOLO [192] et SSD [150] où l'entraînement en détection du détecteur est fait d'une seule traite. Ce réseau est non seulement bien plus rapide

en inférence et en entraînement, mais a aussi de meilleurs performances en détection, car le CNN est optimisé directement pour la bonne tâche. Ce qui rend inutile les analyses difficiles sur les différents fine-tuning à réaliser sur les différentes parties du réseau jusqu'alors indépendantes.

4.1.3 Faster R-CNN [196]

Mais si les CNNs sont si bons en détection d'objets pourquoi se reposer sur la recherche sélective ? Ne serait-il pas possible d'apprendre l'étape de propositions de régions en même temps que celle de classification ? C'est la question à laquelle répond Faster R-CNN en donnant naissance aux RPN (Region Proposal Networks). Les régions sont apprises et inférées de manière complètement convolutionnelle en même temps que la classification fine des meilleurs régions courantes *sur le même CNN* par le mécanisme de RoI pooling évoqué à la section précédente.

Les régions sont apprises en utilisant le système d'ancres vu avec SSD. Pour chaque pixel du descripteur carte, plusieurs a priori sont définis. Une régression à partir de chacun de ces a priori (analogie avec l'apprentissage résiduel) est apprise ainsi qu'un score d'objet pour chaque ancre. Les régions ainsi déformées passent par une étape de suppression des Non Maxima.

Il y a cependant une distinction pour apprendre la classification fine : *les régions proposées par le réseau sont considérées comme fixes alors que leurs positions dépendent aussi des poids des premières couches utilisées pour la classification.* La raison est que le gradient du RoI pooling par rapport à la région n'est pas bien défini à cause de la quantification et des maximas durs qui sont calculés (ainsi qu'à cause de l'étape de Non-Maximum Suppression). Dans un framework de Deep Learning classique cela se traduit par des gradients nuls : la contribution des descripteurs cartes en termes de positions et de sélection des régions n'est pas prise en compte pour la classification/régression.

$$\begin{aligned} \frac{\partial RoIpool(f,(x,y,w,h))}{\partial(x,y,w,h)} &= ? \rightarrow 0 \\ \frac{\partial RoIpool(f,(x,y,w,h))}{\partial f} &= \frac{\partial f}{\partial f}_{indicesdesmaximas} = 1_{indicesdesmaximas} \end{aligned} \tag{4.1}$$

Ce système de détection est maintenant devenu l'équivalent de ce qu'était le DPM en son temps. La quasi-intégralité des contributions du challenge PASCAL-VOC ou MS-COCO sont basées sur des algorithmes qui ne sont fondamentalement que des modifications (parfois très intéressantes) de ce framework pour pallier à tel ou tel problème. Cette fin de chapitre est une tentative d'intégrer l'invariance à la rotation dans ce framework, comme vu avec la cascade dans la partie d'estimation d'angles.

Comme améliorations notables orthogonales à notre travail, nous pouvons citer dans le désordre :

— Le RoI-pooling dépendant de la position [41] dans lequel au sein d'une carte convolutionnelle de profondeur d chaque couche est responsable d'une seule des $N_y * N_x$ cellules du RoI pooling,
— HyperNet [132] évoqué précédemment utilise plusieurs cartes convolutionnelles à différentes résolutions comme avec SSD pour prédire les ancres,
— La pyramide de caractéristiques [144] dans laquelle différentes cartes convolutionnelles du réseau sont utilisées pour prédire chacune des ancres de différentes tailles et sont liées par des connexions de haut en bas (classique), mais aussi de bas en haut ainsi que latérales (en utilisant l'interpolation bilinéaire pour rendre les dimensions spatiales des couches compatibles entre elles),
— Le RoI pooling déformable [42, 166] où une branche du réseau prédit des décalages sur les centres de chaque cellule du RoI pooling,
— Les convolutions déformables [42], où une branche du réseau prédit des décalages sur les positions des composants des filtres de convolutions,
— Le RoI align [87] dont nous reparlerons à la fin de ce chapitre,
— La prédiction de masque en parallèle [87], qui améliore la détection en entraînant simultanément le réseau en segmentation sémantique,
— La stratégie de hard-mining OHEM [215], vue précédemment,
— Le coût focal [145], qui bien qu'appliqué à une autre architecture dans le papier pourrait être aussi bien intégré dans Faster R-CNN,
— Les ancres méta [255] : la fonction de déformation d'ancres ne déforme plus un nombre fixe d'a priori, mais est une fonction continue de l'ancre, qui peut être

choisie optimalement par le réseau pendant l'entraînement, ou par l'utilisateur en phase de test,

— Le RoI pooling précis [121] permettant d'avoir un RoI pooling différentiable par rapport aux coordonnées des régions en plus d'un alignement parfait.

Cette liste est non exhaustive et n'inclut que les modifications majeures. Les résultats de MS-COCO en détection cette année n'étant pas sortis nous nous permettons de prédire (sans beaucoup de risque) que le gagnant dévoilé à ECCV 2018 utilisera une variante de Faster R-CNN.

Maintenant que le lecteur a une complète compréhension de Faster R-CNN, nous allons pouvoir passer à la contribution principale de ce chapitre, à savoir Faster RER-CNN.

4.2 Détecteur à deux niveaux avec ancres tournantes

Il s'agit dans cette section de développer notre contribution en intégrant dans Faster R-CNN, que nous venons de présenter, des boîtes tournantes ainsi que de la régression d'orientation. Le tout afin d'observer les mêmes gains, que nous avions obtenus dans le chapitre précédent table 3.5, mais sur une architecture plus forte et plus élégante. Si l'idée de base est simple, sa réalisation est cependant moins aisée pour les raisons suivantes :

— il faut une paramétrisation de boîte tournante robuste,
— il faut une étape de sélection des positifs basés sur des positions de boîtes tournantes relatives,
— il faut un NMS tournant,
— il faut un RoI pooling tournant,
— il faut une régression de boîtes tournantes non ambiguë.

Toutes ces étapes ont été réalisées et combinées avec succès au sein de la thèse et seront développées dans la section suivante.

4.2.1 Construction des différents blocs de notre détecteur

Paramétriser des boîtes orientées

La paramétrisation bien connue d'une boîte orientée est :

$$\mathcal{B} = (x_{min}, y_{min}, x_{max}, y_{max}), \tag{4.2}$$

ce qui correspond aux coins supérieur gauche et inférieur droit en coordonnées image. Bien sûr, si nous permettons à la boîte de tourner, cela ajoute des paramètres. Nous voulions trouver la paramétrisation la plus simple et la plus proche de cette dernière pour que la transition du système classique au nôtre soit plus aisée. Nous étions aussi motivés par le besoin d'unifier les annotations trouvées dans les BDD d'imagerie aérienne en un format proche de celui de Pascal-VOC. Par exemple les annotations de Joao et Vedaldi dans [95] pour la BDD GoogleEarth [94], sont un parfait exemple du fait que les auteurs ne considèrent en général pas la détection et l'estimation d'angle comme deux problèmes intimement liés. Les annotations de VeDAI peuvent aussi être porteuses de confusions, car certaines sont dans $[0, \pi]$ et d'autres dans $[-\pi, \pi]$.

Nous avons choisi la convention suivante :

$$\mathcal{B} = (x_A, y_A, x_C, y_C, \theta), \tag{4.3}$$

Dans laquelle A est un des coins de la boite (A peut être choisi sur n'importe quel coin *tant que ce choix reste cohérent dans la BDD*) et B, C et D les autres coins pris dans le sens horaire (et donc anti-horaire dans le plan images). Dans les annotations de la base VeDAI, A est choisi comme étant le coin arrière gauche des voitures orientées, de sorte que AB et CD soient les deux côtés les plus longs. Il est crucial que A, B, C et D soient dans cet ordre. Nous verrons pourquoi plus tard. Cette paramétrisation englobe tous les cas d'utilisation trouvés en imagerie aérienne et impose un a priori dur sur la structure des véhicules qui sera utilisée pour la classification.

Ce système a aussi la bonne propriété d'être exactement le même que celui de Faster R-CNN si et seulement si $\theta = 0$. Il est donc complètement différent de [214], qui utilise

w et h à la place de x_C et y_C et encore plus du système complexe à 10 coordonnées de [151], qui donne certainement plus de flexibilité pour la tâche de détection de texte, mais qui n'est pas utile pour nous. Nous modifions les annotations de tous les datasets utilisés pour les rendre au format PASCAL-VOC (pouvant donc être utilisés par Faster R-CNN) avec les 5 champs supplémentaires :

— (xA,yA) et (xC, yC)

— l'angle en radians entre AB et l'horizontale

Ainsi les vérités terrain peuvent être chargées sous la forme d'un tableau 2D `(image_index_in_batch,[xA, yA, xC, yC, theta])`.

IoU avec des rectangles orientés

L'IoU n'est pas seulement un moyen de vérifier la compatibilité entre une boîte prédite et la vérité terrain, mais aussi une étape importante dans tous les pipelines modernes de détection (pour vérifier si plusieurs boîtes peuvent être fusionnées par exemple). Nous voulions un algorithme capable de garder la simplicité de celui existant tout en étant assez rapide pour nos besoins. Pour cela, nous commençons par calculer la longueur et largeur du rectangle. Cela nous donne :

$$\begin{aligned} AB &= \cos(\theta) * \Delta_x + \sin(\theta) * \Delta_y \\ BC &= -\sin(\theta) * \Delta_x + \cos(\theta) * \Delta_y \end{aligned} \tag{4.4}$$

Avec : $\Delta_x = x_C - x_A$ et $\Delta_y = y_C - y_A$. Le grand avantage de cette paramétrisation est qu' AB et BC sont toujours positifs quel que soit l'angle et donc nous n'avons pas besoin de prendre des valeurs absolues. Maintenant nous avons donc accès aux aires des rectangles pris séparément. Pour l'intersection, c'est plus complexe. Comme tous les rectangles sont labellisés dans l'ordre horaire, nous pouvons utiliser directement l'algorithme de Sutherland Hogman [52] pour trouver l'intersection de polygones convexes. Il suffit de clipper un polygone successivement en utilisant tous les côtés de l'autre. En notant subjectPolygon la liste de paires de coordonnées à clipper et clipPolygon la liste de paires de coordonnées du polygone clippant, cela donne le pseudocode de l'algorithme 1.

L'Annexe D contient une version plus bas niveau proche d'un code Python avec les

Algorithm 1 Sutherland-Hogman

Require: subjectPolygon ∨ clipPolygon
  for *clipEdge in clipPolygon* **do**
    $outputPolygon \leftarrow subjectPolygon$
    $inputPolygon \leftarrow outputPolygon$
    $outputPolygon \leftarrow \emptyset$
    $S \leftarrow dernier\ coin\ de\ inputPolygon$
    for E *in* $inputPolygon$ **do**
      if E *du bon cote de clipEdge* **then**
        if S *pas du bon cote de clipEdge* **then**
          $outputPolygon+ = clipEdge \cap [SE]$
        end if
        $outputPolygon+ = E$
      else if S *du bon cote de clipEdge* **then**
        $outputPolygon+ = clipEdge \cap [SE]$
      end if
      $S = E$
    end for
  end for

optimisations réalisées.

Nous présentons un exemple de l'algorithme sur un W clippé successivement par un polygone convexe Figure 4.3. Après avoir obtenu l'intersection (si elle n'est pas vide dans le cas où le nombre de points est inférieur à 3), comme le résultat est aussi en sens horaire, il suffit d'appliquer la formule de Green-Riemann des polygones pour calculer son aire A. Soit une courbe plane C positivement orientée et C^1 par morceaux, D le compact compris à l'intérieur. Si les dérivées des fonctions P et Q sont suffisamment régulières sur D, on a d'après Green-Riemann :

$$\int_C P\,\mathrm{d}x + Q\,\mathrm{d}y = \iint_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) \mathrm{d}x\mathrm{d}y. \tag{4.5}$$

Supposons que nous trouvions P et Q tels que $\left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) = 1$, nous aurions alors :

$$A = \iint_D 1\mathrm{d}x\mathrm{d}y. = \int_C P\,\mathrm{d}x + Q\,\mathrm{d}y \tag{4.6}$$

C'est chose faite en prenant par exemple $Q(x, y) = x$ et $P(x, y) = 0$, ce qui donne après

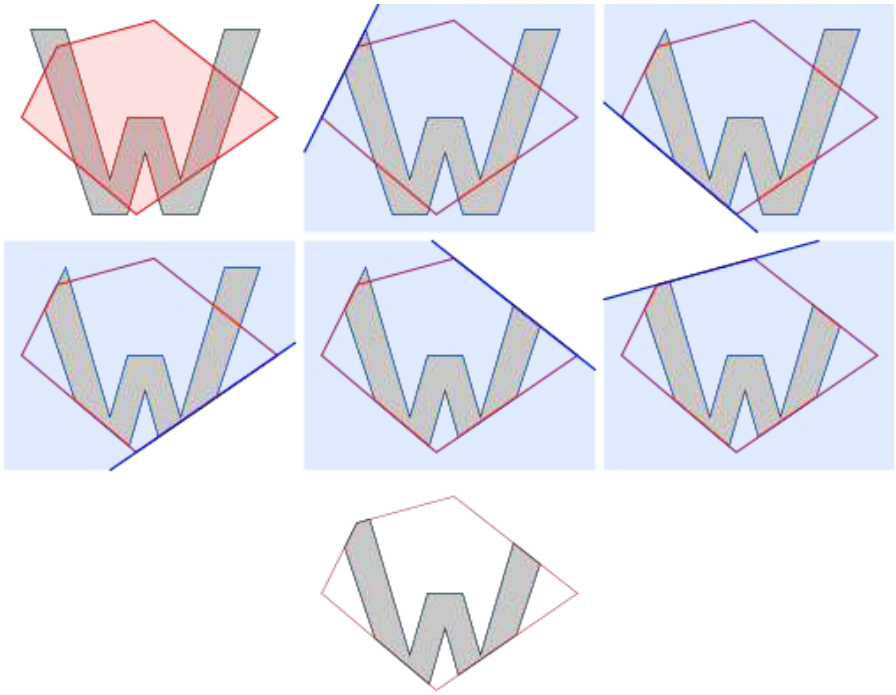


FIGURE 4.3 – Sutherland-Hogman en détails sur le clippage d'un polygone (source wikipedia)

intégration la formule :

$$A = \frac{1}{2}\left|\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right| \tag{4.7}$$

Avec la formule pour l'aire de l'intersection ci-dessus, nous pouvons calculer l'IoU de manière simple. Ceci est résumé dans la Figure 4.4, qui montre le résultat de la formule de GR sur l'intersection de sommets notés de I_1 à I_6. Maintenant que nous disposons des fonctions de base pour calculer l'aire de l'intersection, nous pouvons écrire les deux fonctions qui vont nous servir dans le code :

— le calcul de l'IoU de toutes les ancres avec toutes les vérités terrain pour réaliser les affectations des positifs et négatifs pour l'entraînement `rotated_bbox_overlaps(array_gt,array_anchors)`

— le NMS appliqué à une liste de boîtes tournées ordonnées `rotated_bbox_nms`

Deux implémentations très optimisées ont été écrites en Cython multithreadé. Chaque structure est typée (les polygones sont des vecteurs de paires, les indices, des entiers, etc.). De plus, il faut désactiver le Global Interpreter Lock (GIL) pour permettre la parallélisation ce qui implique de faire du code pur C. Nous avons optimisé de plus

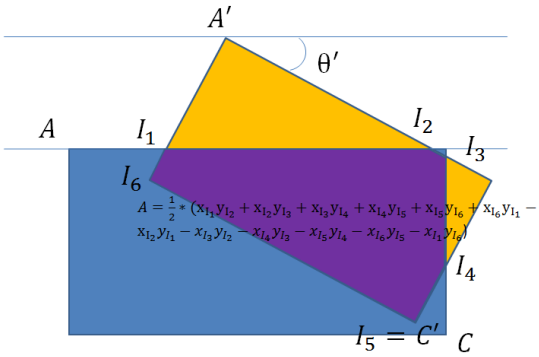


FIGURE 4.4 – Calculer l'aire de l'intersection de deux rectangles orientés après application de Sutherland-Hogman

Sutherland-Hogman en faisant des regroupements non présents dans l'algorithme de Rosetta-code [164].

Le code a besoin d'être très rapide pour ne pas ralentir l'entraînement et l'inférence. C'est pourquoi [151] préfère des simulations Monte-Carlo. Notre algorithme de calcul d'IoU a l'avantage d'être exact, car aucune approximation n'est réalisée, et d'avoir une vitesse d'exécution bonne si la fréquence des CPUs disponibles n'est pas trop faible.

Notre algorithme parcourt toutes les prédictions et les boites vérités terrain et calcule l'IoU de toutes les boîtes entre elles. Pour le NMS les deux boucles ne sont pas totalement parallélisables, mais nous parallélisons la boucle la plus intérieure. Un test du calcul d'IoU est réalisé en dupliquant une boîte tournée et en faisant tourner sa copie autour de son centre ou en lui faisant effectuer une translation horizontale. Les résultats sont affichés ci-après Figure 4.5.

RoI pooling orienté (v1)

Augmenter le RoI pooling n'est pas complètement nouveau [87] et d'autres pooling warping ont déjà été proposés [118, 40, 155]. Nous préférons la simplicité plutôt que la précision et choisissons une implémentation très proche de l'original non orienté [70]. Cela veut dire que nous utilisons la même quantification dure pour définir les coins des cellules de pooling, et implique que les cellules résultantes ne sont pas forcément rectangulaires. Ensuite, nous passons en revue tous les pixels du descripteur carte contenus

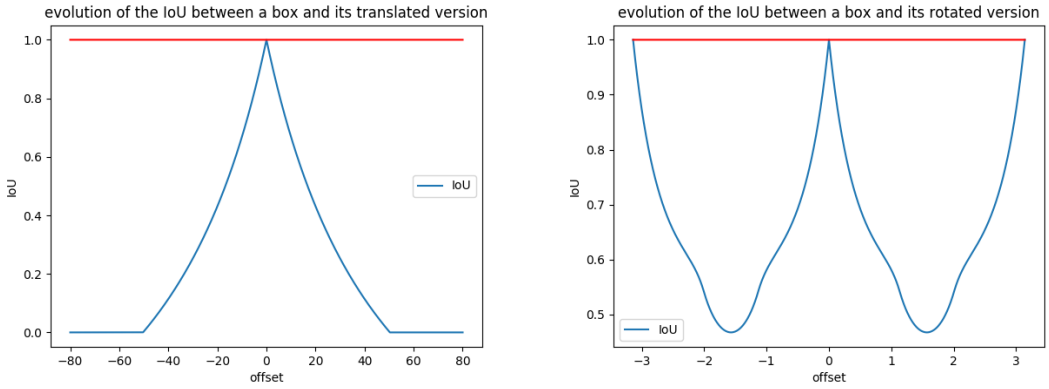


FIGURE 4.5 – À gauche nous dupliquons une boîte et nous effectuons une translation de sa copie en mesurant son IoU à l'original, à droite, nous faisons tourner sa copie autour de son centre

dans le rectangle vertical englobant de la cellule courante et vérifions leur appartenance en utilisant des critères à base de produits scalaires/produits mixtes. Nous prenons ensuite le maximum de tous les pixels respectant le critère. Ceci est représenté Figure 4.8. Pour la backward nous stockons les indices des maximas dans leur cellule respective de pooling et nous parcourons la carte convolutionnelle en ne considérant que les pixels dans le rectangle vertical englobant puis seulement les pixels dans le quadrilatère, en testant pour chaque pixel si son indice est le même que celui stocké.

Nous montrons un test visuel de notre opération sur la Figure 4.7 pour une région 20x20 : la quantification est très brutale et nous pouvons observer que la voiture résultante est déformée dans une certaine mesure à cause des approximations utilisées dans les cellules de pooling. Ces implémentations sont réalisées en C++ multi-threadé et en CUDA pour être utilisées comme nouveau module par le framework Tensorflow [2]. Il faut implémenter la forward, qui est simplement l'opération de calcul de maximas ainsi que la backward qui transforme le gradient des couches supérieures. Les principales difficultés des implémentations sont qu'il faut aplatir tous les tableaux afin de partager le remplissage entre les threads/warps pour le GPU. Il y a donc un constant travail de passage des indices aplatis aux indices dans les tableaux 4D et réciproquement. De plus, si plusieurs régions se recouvrent, le même pixel est donc utilisé plusieurs fois. Il contribue donc d'autant plus au gradient. Pour vérifier la qualité de notre implémentation, autrement que par des tests unitaires sur des exemples jouets, il est nécessaire de vérifier si

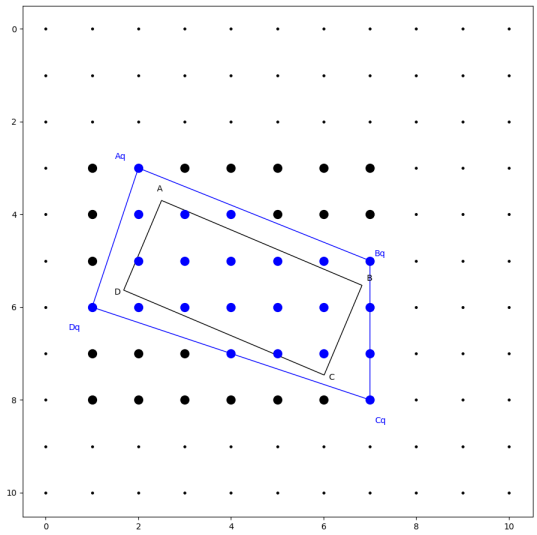


FIGURE 4.6 – RoI pooling tournant v1.0 : La grille des centres des pixels est en noir, avec en petit les pixels non considérés, car n'étant pas dans le rectangle vertical englobant, et en gros les pixels considérés. Les pixels finalement utilisés pour le maximum sont en bleus.

notre gradient analytique est le bon. Nous comparons donc pour des matrices aléatoires la différence entre l'approximation centrée des différences finies et le gradient analytique sur la jacobienne toute entière de dimension $(N * N_y * N_x * D)$ par $(BATCH * H * W * D)$.

$$\frac{f_i(x + 2 * h * I(j)) - f_i(x - 2 * h * I(j))}{2 * h} \approx \frac{\partial f_i(x)}{\partial x_j} \tag{4.8}$$

Avec $I(j)$ le vecteur nul sauf en j ou il vaut 1. Bien évidemment le gradient vaut partout

FIGURE 4.7 – De gauche à droite : une région d'intérêt et sa paramétrisation dans l'image de départ, les emplacements des pixels des maximas poolés dans l'image de départ (argmaxs)(backward), les pixels poolés réordonnés par le RoI pooling tournant (on utilise l'image d'origine pour une visualisation plus compréhensible)(forward)

0 sauf en les pixels utilisés pour le pooling de la cellule correspondante ou il vaut 1. Il existe beaucoup de littérature sur le choix de h ainsi que l'écart acceptable. Pour le mesurer de manière satisfaisante il faut implémenter aussi les noyaux de calculs en doubles (les simples (flottants 32 bits) étant utilisés par défaut dans Tensorflow). Notre test passe à 10^{-13} près, [124] recommandant 10^{-7}.

Addendum : Ceci est notre première version de l'algorithme. La version 2 "Align" est développée dans la section suivante. Il est tout à fait possible de trouver d'autres formulations de la v1 se justifiant tout autant. Nous aurions pu, par exemple, supprimer la quantification des coins des cellules de pooling, qui ne sert en fait absolument à rien (à part à retrouver le résultat du RoI pooling de Girshick), et qui, en plus, introduit des déformations de cellules provoquant un décalage supplémentaire entre l'endroit poolé en théorie et le maximum effectivement utilisé. Les critères ne seraient plus alors des produits mixtes, mais des produits scalaires. Les deux variantes possibles de la v1 sont résumées dans l'algorithme 2.

Régression des boîtes orientées

La partie vitale de l'implémentation est la régression de boîtes tournantes. Par exemple si nous régressons directement le décalage (Δ_θ) entre l'angle prédit θ et la cible θ_t en radian, nous courrons au-devant de plusieurs problèmes. En effet, en utilisant la paramétrisation simple décrite dans la section 4.2.1 nous avons introduit beaucoup d'ambiguïté car il y a maintenant 4 manières valides de placer A même pour une prédiction parfaite au sens géométrique (rectangle prédit=rectangle vrai). Il y a donc 4 Δ_θ associés. Il faut regarder chaque cas séparément. Nous utilisons l'indice t pour la cible :

- A est près de (ou sur) $A_t \Rightarrow \Delta_\theta = 0$, $AB \approx A_tB_t$, $BC \approx B_tC_t$
- A est près de (ou sur) $B_t \Rightarrow \Delta_\theta = \frac{\pi}{2}$, $AB \approx B_tC_t$, $BC \approx A_tB_t$
- A est près de (ou sur) $C_t \Rightarrow \Delta_\theta = \pi$, $AB \approx A_tB_t$, $BC \approx B_tC_t$
- A est près de (ou sur) $D_t \Rightarrow \Delta_\theta = \frac{3\pi}{2}$, $AB \approx B_tC_t$, $BC \approx A_tB_t$

Nous voulons du gradient dans le deuxième et quatrième cas. Autrement, nous nous éloignerions de la vérité terrain quand même en régressant AB sur B_tC_t, etc. Dans les

Algorithm 2 RoI pooling tournant pour une cellule

Require: `input_tensor` $\vee$ `roi_cell`$= (xA, yA, xB, yB, xC, yC, xD, yD)$
  $xAq \leftarrow FLOOR(xA)$
  $yAq \leftarrow FLOOR(yA)$
  $xBq \leftarrow CEIL(xB)$
  $yBq \leftarrow FLOOR(yB)$
  $xCq \leftarrow CEIL(xC)$
  $yCq \leftarrow CEIL(yC)$
  $xDq \leftarrow FLOOR(xD)$
  $yDq \leftarrow CEIL(yD)$
  $ymax \leftarrow MAX([yAq, yBq, yCq, yDq])$
  $ymin \leftarrow MIN([yAq, yBq, yCq, yDq])$
  $xmax \leftarrow MAX([xAq, xBq, xCq, xDq])$
  $xmin \leftarrow MIN([xAq, xBq, xCq, xDq])$
  if `version 1 alternative` **then**
    $AB2 = (xB - xA) * (xB - xA) + (yB - yA) * (yB - yA)$
    $BC2 = (xB - xC) * (xB - xC) + (yB - yC) * (yB - yC)$
  end if
  `max_value`$\leftarrow -\infty$
  for $(xg, yg) \in$`input_tensor` **do**
    if $xg \geq xmin \vee xg \leq xmax \vee yg \leq ymax \vee yg \geq ymin$ **then**
      if `version 1 alternative` **then**
        `inside_AB`$= (0 \leq (xg - xA) * (xB - xA) + (yg - yA) * (yB - yA) \leq AB2)$
        `inside_BC`$= (0 \leq (xg - xB) * (xC - xB) + (yg - yB) * (yC - yB) \leq BC2)$
        inside =`inside_AB`$\vee$ `inside_BC`
      end if
    else if `version 1` **then**
      `inside_AB`$= (0 \leq (yg - yAq) * (xBq - xAq) - (xg - xAq) * (yBq - yAq))$
      `inside_BC`$= (0 \leq (yg - yBq) * (xCq - xBq) - (xg - xBq) * (yCq - yBq))$
      `inside_CD`$= (0 \leq (yg - yCq) * (xDq - xCq) - (xg - xCq) * (yDq - yCq))$
      `inside_DA`$= (0 \leq (yg - yDq) * (xAq - xDq) - (xg - xDq) * (yAq - yDq))$
      inside =`inside_AB`$\vee$`inside_BC`$\vee$`inside_CD`$\vee$ `inside_DA`
    end if
    if *inside* **then**
      if `input_tensor`$(xg, yg) >$`max_value` **then**
        `max_value`=`input_tensor`(xg, yg)
        `output_tensor`(`index_cell`)=`max_value`
        `output_argmax`(`index_cell`)$= (xg, yg)$
      else
        continue
      end if
    else
      continue
    end if
  end for

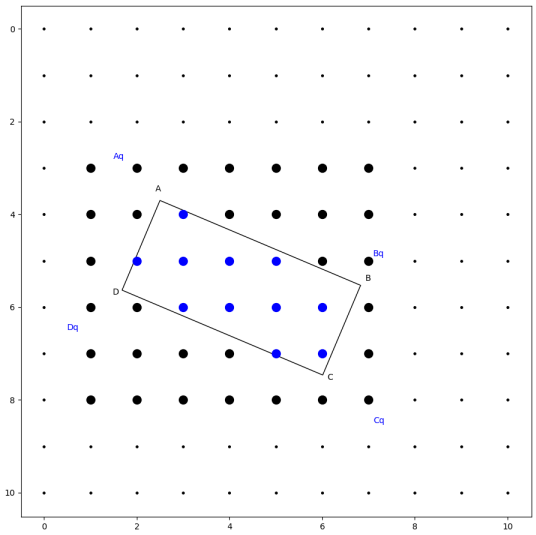


FIGURE 4.8 – RoI pooling tournant v1 alternative : La grille des centres des pixels est en noir, avec en petit les pixels non considérés, car n'étant pas dans le rectangle vertical englobant, et en gros les pixels considérés. Les pixels finalement utilisés pour le maximum sont en bleus.

autres cas, nous ne voulons pas de gradient, car le fait de forcer un côté arbitraire de l'ancre sur un côté particulier de la boîte nuit à la généralisation et n'apporte rien en termes d'IoU. Bien sûr il y a aussi le problème de la $2 * \pi$ périodicité. Utiliser le coût ci-dessous résout tous les problèmes :

$$\begin{aligned} \Delta_\theta &= \theta_t - \theta, \\ v &= [\Delta_\theta - \pi, \Delta_\theta + \pi, \Delta_\theta], \\ index &= \arg\min(|v|), \\ \Delta_\theta &= v[index] \end{aligned} \tag{4.9}$$

On peut ensuite juste ajouter ce terme de régression d'angle Δ_θ aux autres termes présents dans la régression de Faster R-CNN c'est-à-dire avec une régression L2 normalisée (et parfois log-normalisée pour les longueurs et largeurs). On présente ici la fonction

de coût de régression complète utilisée :

$$
\begin{aligned}
RPN = \sum_{a,t} p * \Bigg(& ||\gamma_1 * (\left(\frac{x_{ct} - x_{ca}}{\frac{W_a + Ha}{2}} \right) - \Delta_x)||_\delta^2 \\
& + ||\gamma_2 * (\left(\frac{y_{ct} - y_{ca}}{\frac{W_a + Ha}{2}} \right) - \Delta_y)||_\delta^2 + ||\gamma_3 * (\log \frac{W_t}{W_a} - \Delta_{\log w})||_\delta^2 \\
& + ||\gamma_4 * (\log \frac{H_t}{H_a} - \Delta_{\log H})||_\delta^2 + ||\gamma_5 * ((\theta_t - \theta_a) - \Delta_\theta)||_\delta^2 \Bigg)
\end{aligned}
\tag{4.10}
$$

Où $||.||_\delta^2$ et le coût traditionnel de Huber (smooth L1) avec paramètre δ. t réfère à la cible a à l'ancre ; l'index c marque le centre ; W est AB et H est BC et les autres lettres grecques non référencées sont des hyper paramètres. Nous intégrons ce coût dans notre nouveau framework Faster R-CNN à boîtes orientées : Faster RER-CNN.

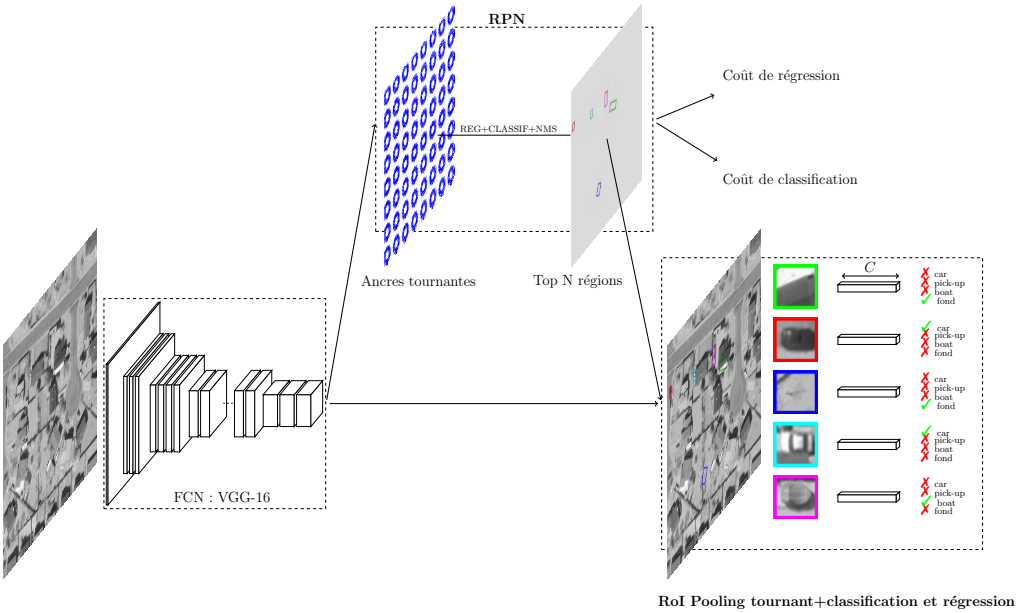
Les ancres

Pour les ancres de base nous ajoutons un paramètre dans la configuration donnant le nombre de duplications des ancres verticales de manière uniforme pour N angles entre $[0, \frac{\pi}{2}]$. L'utilisateur peut aussi décider de privilégier certaines orientations et peut choisir des ancres à la main si il existe des biais d'angles dans la BDD, mais nous ne recommandons pas ce réglage en pratique. Pour plus de détails, les ancres seront étudiées plus en détail section 4.2.2.

Nous disposons maintenant de toutes les briques nécessaires, le reste est de l'ingénierie pure et dure. L'ensemble du processus est représenté Figure 4.9.

4.2.2 Validation experimentale

En plus de la base VeDAI [189] sur laquelle nous testons d'ailleurs le multi-classes comme avec SSD et YOLOv1 nous utilisons aussi Munich [141] pour vérifier que notre framework s'adapte à tous les contextes d'imagerie aérienne. Les images de Munich doivent être découpées pour un apprentissage efficace (et parce qu'elles ne rentreraient pas dans la mémoire RAM du GPU). Nous coupons les grandes images en larges patchs 512x512 non recouvrants que nous ré-échantillonnons en 1024x1024 en utilisant de l'interpolation bilinéaire. En faisant cela nous avons manqué des cibles. Nous échantillonnons des



1

FIGURE 4.9 – Faster RER-CNN

fenêtres 512x512 additionnelles, de telle manière que chaque vérité terrain soit trouvée au moins une fois dans une tuile. Pour le Test, nous effectuons un parcours par fenêtre glissante de pas 50 pixels comme dans [230], et en re-échantillonnant chaque tuile en 1024x1024. Le ré-échantillonnage est nécessaire pour que le pooling ne dégrade pas les performances comme dans [261].

Finalement, nous utilisons aussi la base Google Earth [94]. Nous testons sur 5 images en entraînant sur 10 (les annotations de Joao et Vedaldi [95] ne sont disponibles que sur 15 images).

Nous utilisons la descente de gradient stochastique (SGD) avec un taux d'apprentissage de 0.001 et un moment non nul. Pour Faster R-CNN c'est en fait trop brutal pour quelques folds et ne converge pas. Nous avons donc utilisé 0.0001 sur les folds en question. Pour Faster R-CNN nous avons aussi utilisé 36 ancres [2] pour vérifier que

2. Nous utilisons un échantillonnage dense de rapports longueur/largeur et de tailles (6 et 6 respectivement)

notre avantage sur Faster R-CNN n'est pas attribuable au nombre d'ancres utilisées par pixel. Nous avons utilisé en majeure partie l'architecture VGG16 [217] pré-entraînée sur IMAGENET.

Toutes les expériences du début de cette section sont conduites sur la fold 1 de VeDAI avec l'objectif de valider que tout fonctionne comme prévu et de produire une analyse complète de notre algorithme. Dans la suite nous utiliserons, bien sûr, toute la BDD pour les comparaisons avec l'état de l'art.

Note sur le protocole d'évaluation. Pour comparer Faster R-CNN et Faster RER-CNN sans favoriser l'un ou l'autre il faut se poser la question de la métrique. Si nous prenons l'IoU de Pascal VOC, nous ne pouvons utiliser les vérités terrain de l'un ou l'autre, car cela ne serait pas équitable pour l'autre, même si les vérités terrain sont les boîtes orientées. Car chacun est entraîné sur un jeu différent de vérités terrain, il est par exemple beaucoup plus dur d'avoir un bon IoU avec une boîte tournée qu'avec une boîte alignée avec les axes compte tenu du degré de liberté supplémentaire. Nous utilisons donc aussi la métrique de VeDAI sur les centres des boîtes (indépendants de leurs orientations) avec laquelle le lecteur est déjà familiarisé.

Comparaison du RPN : avec ou sans ancres orientées

Dans ces expériences nous voulons démontrer que même à ce niveau-là il est important d'avoir des ancres orientées (au second niveau, cela a déjà été démontré dans le chapitre précédent). Pour le montrer, nous mesurons, sur un ensemble de validation, le rappel moyen sur les classes par rapport à la métrique VeDAI (voir Section 4.2.2) en fonction du nombre de propositions retenues. Nous extrayons les N meilleures propositions du RPN avant régression : là où les centres sont donc exactement les mêmes dans Faster R-CNN et sa contrepartie tournante (ils sont situés sur la même grille), et après *régression* quand les propositions peuvent bouger et se dilater. Bien sûr, notre framework offre bien plus de flexibilité. Il a été construit pour ça. En regardant la Figure 4.10 l'avantage qu'apportent les ancres tournantes est clair. Ce qui est surprenant est que, même pour un faible nombre de propositions (400), l'écart entre les deux RPNs est

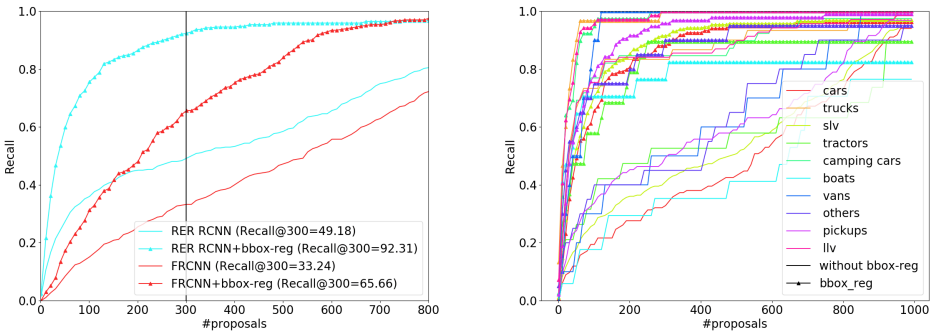


FIGURE 4.10 – À gauche nous comparons le rappel moyenné sur toutes les classes en fonction du nombre de propositions avant et après régression pour les deux frameworks. A droite nous présentons le détail par classe

vraiment marqué : de 65.66 à 92.31 (une augmentation de 41% !) après régression. Plus surprenant encore est le fait que *même avant la régression le rappel est plus haut* ! (de 48% !) comme la métrique n'est fondée que sur les centres et qu'ils sont les mêmes cela veut donc dire que de meilleures ancres sont choisies, car l'apprentissage a été rendu plus facile ou plus général par notre formulation. La partie droite de la même Figure 4.10 donne le détail par classe.

Régression d'angle dans la tête de classification

Maintenant que nous avons montré que notre RPN est fonctionnel. Nous voulons vérifier que la régression d'angle de Fast R-CNN et du RPN est bonne (même si les résultats précédents semblent indiquer que c'est bien le cas au moins pour le RPN). Nous montrons un histogramme de la répartition de l'erreur absolue d'angle entre les détections finales, vraies positives, et leur vérité terrain correspondante.

Seul un seul patch est détecté avec plus de 25 degrés d'erreur et la plupart des erreurs sont négligeables (< 10 degrés). Nous notons que Liu and Mattyus [149] ont aussi de bons résultats, mais notre framework est plus simple et plus élégant car end-to-end. La seule vraie erreur (> 40 degrés) vient d'une vérité terrain qui semble fausse (visible sur la Figure 4.11 aussi).

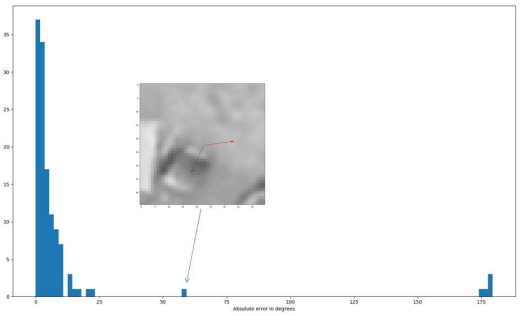


FIGURE 4.11 – Histogramme des erreurs angulaires entre les bonnes prédictions et les vérités terrain associées

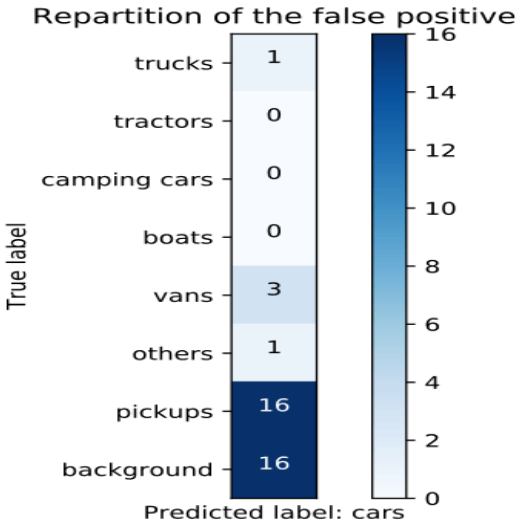


FIGURE 4.12 – Vecteur de confusion de la classe voiture

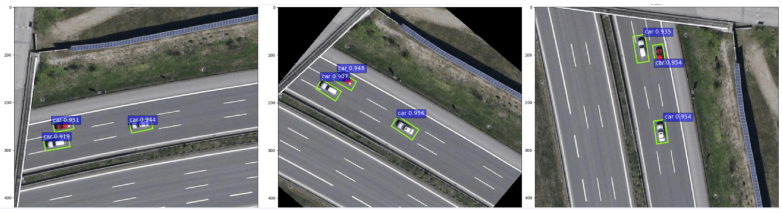


FIGURE 4.13 – Détections de notre Faster RER-CNN en fonction de la rotation de l'image

Equivariance en rotation des propositions

Nous réalisons des expériences additionnelles sur le test de Munich pour vérifier les performances de notre détecteur sur des séries d'images tournées et regardons comment évoluent le rappel et la précision. Grâce à notre implémentation, notre framework est stable pour différentes versions d'images tournées à différents angles. Nous présentons un sous-ensemble des détections Figure 4.13. Les expériences montrent que le détecteur est équivariant dans une certaine mesure, ce qui justifie son nom de Faster Rotation Equivariant R-CNN.

Analyse des confusions du détecteur de voiture

Nous nous focalisons maintenant sur le détecteur de voitures extrait de notre algorithme Faster RER CNN entraîné sur VeDAI et affichons les résultats d'une étude

FIGURE 4.14 – Les faux-positifs pour le détecteur de voitures, extrait de Faster RER-CNN, dont le score est supérieur à 0.5. Nous avons utilisé le code-couleur suivant : vans : turquoise, trucks : bleu foncé, pick-up : violet, others : jaune, fonds : rose

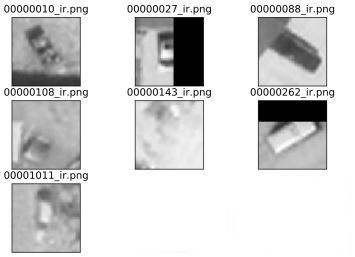


FIGURE 4.15 – Les seules 7 vérités terrains jamais trouvées par notre détecteur de voiture (sur 121 dans la première fold) : les patchs 4 et 5 ne ressemblent pas à des voitures et sont probablement des erreurs d'annotation et les vignettes 2,6 et 7 présentent des véhicules occultés.

poussée sur les voitures non trouvées et sur les faux positifs. En ceci nous nous inspirons du travail de [103] qui aide à diagnostiquer les erreurs des détecteurs. Les résultats de cette étude se trouvent Figure 4.12, où nous pouvons voir la répartition de tous les faux positifs dont le score est supérieur à 0.5 (c'est donc un vecteur colonne tiré d'une matrice de confusion). La Figure 4.14 montre les vignettes associées (non-tournées et non-élargies). Il est évident que la plupart des erreurs sur les fonds sont en fait des erreurs de localisation sur des cibles. Nous affichons aussi les voitures qui ne sont jamais trouvées par notre détecteur Figure 4.15.

Il est maintenant temps de regarder les performances de notre détecteur sur différents benchmarks évoqués précédemment.

Résultats sur la base VeDAI

Quelques exemples de détection sur VeDAI peuvent être vus Figure 4.16.

Les courbes précision-rappel associées sont présentées Figure 4.17. La Table 4.6 multi-classes montre qu'un énorme gain en performance est observé sur tous les véhicules (sauf pour les avions et les bateaux où les chiffres sont sensiblement les mêmes) ce qui montre que notre framework est bénéfique. Il y a une forte variance sur quelques classes avec peu d'exemples. La variance inter-fold n'est aussi pas négligeable non plus. Pour plus d'explications sur la confusion entre les classes, nous référons le lecteur à la section 4.2.2.

FIGURE 4.16 – Des exemples de détection sur les 3 BDDs de gauche à droite : Munich3k, VeDAI SII, GoogleEarth

method	AP	Recall@0.01FPPI
DPM [189]	60.5 ± 4.2	13.4 ± 6.8
SVM+HOG31 [189]	55.4 ± 2.6	7.8 ± 5.5
SVM+LBP [189]	51.7 ± 5.2	5.5 ± 2.2
SVM+LTP [189]	60.4 ± 4.0	9.3 ± 3.7
SVM+HOG31+LBP [189]	61.3 ± 3.9	8.3 ± 5.2
SVM Fusion AED (HOG) [188]	69.6 ± 3.4	20.4 ± 6.2
Wide LeNet-5+HM (CV) [170]	77.80 ± 3.3	31.04 ± 11
Faster R-CNN (notre implementation)	77.69 ± 3.4	20.85 ± 14
Faster RER-CNN	$\mathbf{80.2 \pm 3.3}$	$\mathbf{33.2 \pm 14}$

TABLE 4.1 – Comparaisons de l'existant avec Faster RER-CNN sur VeDAI

Résultats sur la base Munich 3K

Sur la base Munich [141], de nombreuses astuces doivent être appliquées pour égaler l'état de l'art, comme utiliser de meilleurs réseaux, OHEM, ou bien augmenter la résolution de la carte convolutionelle sur laquelle nous calculons les ancres. Nous ne sommes pas allés aussi loin. Cela explique les résultats de la Table 4.3.

Nous observons que, comme pour VeDAI, PASCALVOC@0.3 et la métrique VeDAI sont similaires, car la plupart des erreurs sont dues, non à des imprécisions de localisation, mais plutôt à des négatifs difficiles ou des véhicules difficiles à trouver.

Il est intéressant de regarder l'annexe C présentant des exemples de détection pour un seuil fixé.

Résultats sur la base GoogleEarth

Nous avons testé ce petit dataset juste pour montrer la généralité de notre solution à tous les problèmes d'imagerie aérienne. Cependant, le nombre d'images et de véhicules

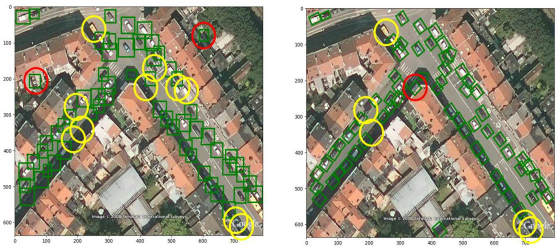


FIGURE 4.18 – Les véhicules ratés par le détecteur sont indiqués avec des cercles jaunes et les faux positifs en rouge. À gauche ; Faster R-CNN. À droite : Faster RER-CNN

est trop faible c'est pourquoi dans les deux cas nous avons probablement sur-appris que ce soit avec Faster R-CNN ou Faster RER-CNN. Cependant même si les différences de performances sont faibles, notre framework mène le jeu comme le montre la Table 4.4. La Figure 4.18 montre, pour l'une des 5 images de test, les résultats des deux framework dont la confiance est supérieure à 0.5.

Comparaison de vitesses

Nous ajoutons Table 4.5 la comparaison de la vitesse de Faster RER-CNN par rapport à Faster R-CNN. le facteur limitant pour notre implémentation est le calcul d'IoU ainsi que le NMS tournant sur CPU (le NMS est sur GPU pour Faster R-CNN) nous nous attendons donc à des ralentissements. Cela prend approximativement deux fois plus longtemps en inférence et en test ce qui est raisonnable, car nous obtenons l'orientation en même temps que le gabarit.

Étude de l'influence des ancres

YOLOv2 [193] propose, pour le choix des ancres, d'utiliser des méthodes plus automatisées (par opposition à choisir les ancres empiriquement) basées sur la répartition des boites vérité-terrain des objets dans l'ensemble d'apprentissage. *En ne prenant pas en compte la largeur du maillage de la grille*, nous regardons l'IoU moyen des vérités terrain à leur ancre la plus proche. Nous essayons de maximiser ce nombre pour un budget d'ancres fixé, ainsi les a priori étant meilleurs il y aura à la fois plus de positifs dans l'apprentissage pour une même taille de grille, et les décalages à régresser seront

FIGURE 4.19 – À gauche toutes les boites vérité-terrain de Munich ramenées au même centre, à droite nous normalisons ces boîtes en angle

moins importants. Du coup le réseau aura moins d'efforts à faire.

Pour cela nous normalisons toutes les vérités terrain en les réunissant sur le même centre et *en leur donnant toutes la même orientation*. Le processus est représenté Figure 4.19. Nous appliquons alors l'algorithme K-means++ [9] avec différents nombres d'ancres. Il est très important d'utiliser une bonne initialisation pour K-means, telle que celle proposée par K-means++ sinon les convergences sont très mauvaises. De plus, il est parfois nécessaire de réaliser l'optimisation plusieurs fois pour être sûr d'avoir le minimum global.

Une fois que nous obtenons ces ancres, nous rétablissons les orientations originales de la vérité terrain et nous dupliquons les angles des ancres à différents angles afin de ne privilégier aucune direction (ce qui n'a pas de sens pour un algorithme général). Ceci est représenté Figure 4.20. Nous représentons alors Figure 4.21 l'IoU moyen obtenu en fonction du nombre d'ancres initiales pour plusieurs valeurs de duplication. La taille des points marque le budget global c'est-à-dire : $nb_{baseanchors} * nb_{duplications}$. Nous voyons d'après l'aspect plat des lignes de mêmes nombres de duplications, qu'il est bien plus intéressant, à budget fixé, d'utiliser plusieurs angles pour un nombre d'aspect-ratios et de taille faible. Sur Munich nous comparons le choix fait main de 36 ancres, avec les ancres obtenues par K-means donnant un IoU de 0.85 pour 36 ancres (9 duplications, 4 ancres de base comme avant, mais choisies par l'algorithme). Cela nous permet de gagner 2 points d'AP sur la base Munich en passant de 84.34 à 86.38.

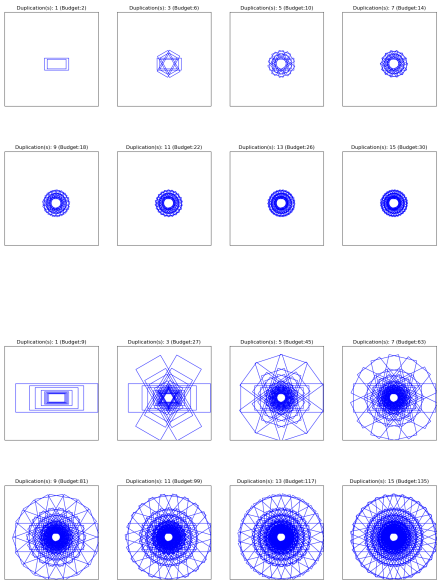


FIGURE 4.20 – Au-dessus les ancres dupliquées pour un budget initial de deux ancres, en bas pour un budget initial de 9 ancres (comme dans l'article original)

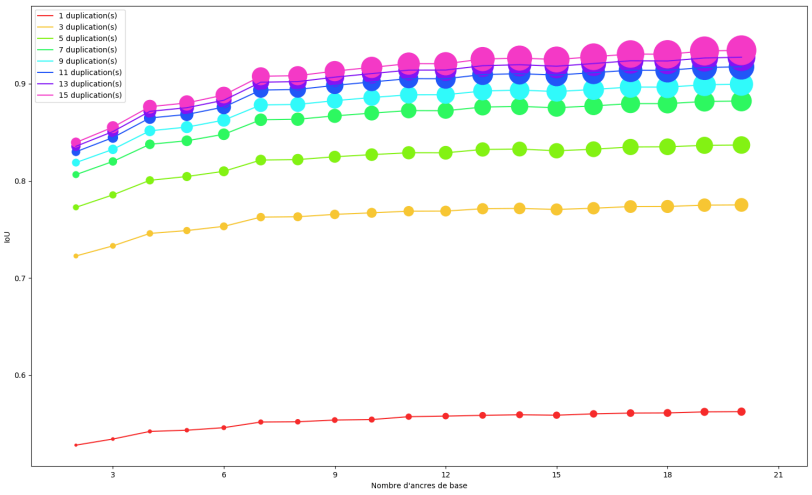


FIGURE 4.21 – Évolution de l'IoU moyen des vérités terrain avec les ancres (ordonnée) en fonction du nombre d'ancres de base (abscisse) et pour différents nombres de duplications

Annexe sur les temps d'expérimentation : L'entraînement d'un Faster RER CNN étant plus long, un seul réseau entraîné jusqu'à convergence sur les 10 folds de VeDAI requiert environ 13 jours. Il n'a donc pas été physiquement possible de pousser très loin les études paramétriques pour les ancres sur VeDAI. *Tous les résultats présentés utilisent des ancres faites-mains.*

4.2.3 Comparaisons avec d'autres travaux similaires

Nous avons eu cette idée d'introduire des ancres tournantes et un RoI pooling tournant en octobre 2016, et avons vraiment commencé à travailler dessus courant 2017, poussés par la sortie sur arXiv en début d'année de [155] proposant une idée similaire et qui a déjà plus de 40 citations. Nous nous différencions de [155], travaux contemporains aux nôtres, sur plusieurs points : notre paramétrisation : [155] utilise (x,y,w,h,θ) avec h le plus petit côté et w le plus grand côté, notre coût de régression prend en compte l'ambiguïté de la paramétrisation dont il fait abstraction en utilisant le plus petit et le plus grand côté, mais ceci rend les calculs lourds et inélégants. Son calcul d'IoU suit les grandes lignes du nôtre, mais les oblige à ranger les points dans le sens horaire au moment de calculer l'aire. Quant à son RoI-pooling tournant il est très différent du nôtre, car il tourne les vignettes alors que nous prenons directement les pixels comme ils viennent grâce à nos critères d'appartenance. Nous nous en éloignerons encore plus dans la version 2. Nous avions fini la v1 en Octobre 2017. Entre temps, beaucoup d'autres articles se sont inspirés de [155] comme [152] qui propose un RoI pooling tournant pour la détection de bateaux qui semble très proche de notre version 1 alternative, bien que peu de détails soient disponibles dans l'article. De plus il ne calcule pas d'IoU entre boîtes orientées, mais régresse à partir des meilleurs ancres horizontales. Nous pouvons aussi citer [20] aussi pour de la détection de texte, qui, à l'aide d'une structure similaire à YOLO, régresse des boîtes orientées. La reconnaissance de texte présente dans leur système, postérieure à la détection, fait intervenir un RoI pooling tournant basé sur l'interpolation bilinéaire pour l'obtention de descripteurs alignés. En ce qui concerne la v2, dont nous parlons dans la section suivante, le RoI Align tournant est aussi présent dans [92].

FIGURE 4.22 – RoI Align tourné avec 4 pastilles par cellules

4.2.4 Mask R-CNN et alignement du RoI pooling : Faster RER CNNv2

Dans un papier très remarqué à ICCV 2017 [87], Girshick modifie encore une fois Faster R-CNN et montre des gains énormes en termes de mAP sur la base MS-COCO en détection. Les deux seules modifications effectuées sont : une branche de segmentation sémantique par classe, qui permet de produire un masque et donc d'obtenir une boîte plus fine, mais qui est inutilisable dans la plupart des datasets d'imagerie aérienne (sauf dans Vaihingen [117]) et une modification du RoI pooling appelé RoI Align. Cette modification vient de la constatation des décalages non négligeables causés par la quantification du RoI pooling. Girshick retire toute quantification en proposant de fixer un nombre fixe de pastilles (4 par exemple) à des coordonnées flottantes uniformément sur chaque cellule (définies aussi par des coins aux coordonnées flottantes non quantifiées) et de réaliser le maximum sur ces pastilles. La valeur des pastilles est obtenue elle-même par interpolation bilinéaire, ce qui, de plus, rend l'opération sous différentiable par rapport aux paramètres définissant la région, tant que les pastilles restent entre les mêmes 4 pixels. [121] propose un RoI pooling complètement différentiable. Cet ajout de pastilles, qui permet de gagner en performance, est en plus trivialement implémentable dans notre version tournée. Les pastilles ne sont plus au même endroit, mais tout le mécanisme reste le même (dans l'implémentation c'est un peu plus complexe). Le résultat est représenté Figure 4.22. Nous implémentons en Tensorflow le RoI Align non tournant ainsi que sa contrepartie tournante. Là encore, les implémentations sont effectuées en C++ et en

CUDA selon les mêmes principes que précédemment. Par contre, nous ne gardons plus l'indice de l'argmax, mais simplement le numéro de la pastille (mais ceci est un choix d'implémentation).

Résultats comparés des deux versions de Faster RER sur VeDAI

Sur VeDAI nous testons les 10 folds comme précédemment en ne modifiant que le RoI pooling tournant. Nous gardons les mêmes initialisations, temps d'apprentissage et architecture. Sur les classes importantes comme voitures, pick-up et camping-cars (d'écart-type faible) nous distinguons une augmentation consistante de 1 à 2 point(s) de mAP. Sur les classes avec moins d'exemples, nous perdons en termes de performances, mais comme les écarts-types sont assez élevés il est difficile de tirer des conclusions. De plus nous n'avons pas effectué de validation croisée sur le nombre d'ancres ou les seuils de NMS pour ce nouveau RoI-pooling. Cela nous permet de compléter notre tableau de résultats sur VeDAI Table 4.7. Nous suggérons au lecteur d'utiliser la version Align du framework, plus mature.

Étude des différents squelettes

Bien entendu, notre pipeline Faster RER-CNN garde toute la flexibilité de son ancêtre en termes d'architectures utilisées. Nous effectuons ici la comparaison entre VGG16 [217], Resnet-50 [90] et l'architecture de FastVeDAI [21], réminiscence de nos premiers travaux sur LeNet-5, sur la base Munich. Pour toutes ces expériences, nous utilisons le RoI align orienté et l'inférence est réalisée comme précédemment sur des images 512x512 sur-échantillonnés d'un facteur 2 espacées de 50 pixels, sauf pour FastVeDAI ou les images ne sont pas sur-échantillonnées (à cause de la limitation de mémoire du GPU). Tous les réseaux sont pré-entraînés sur ImageNet sauf LeNet-5 qui est appris "from scratch", les résultats sont présentés Table 4.8. Deux structures de Resnet50 différentes sont présentées :

— la première conserve les couches de ResNet-50 fixées jusqu'à `res4f_relu` puis après RoI-pooling (tournant ou non tournant aligné ou non aligné) enchaîne les couches de convolutions et complètement connectées sans dropout et avec nor-

malisation de batch (ResNet-50^O),

— La deuxième fait pareil, mais réduit d'abord la dimension de profondeur de `res4f_relu` de 1024 à 512 avant pooling et utilise un RoI pooling 5x5 (ce qui permet de réduire la RAM GPU) pour enchaîner sur les mêmes couches complètement connectées que VGG-16 avec dropout (ResNet-50^J).

Le deuxième ResNet-50^O n'est pas entraîné jusqu'à convergence, car il sur apprend. Mais nous voyons l'apport en termes de représentativité des meilleurs réseaux. La performance de LeNet-5 n'est pas si mauvaise en comparaison, car étant appris "from scratch" et n'ayant pas de stratégie de recherche d'exemples difficiles.

4.3 Conclusions

Nous avons construit dans ce chapitre un second détecteur basé sur l'estimation d'angle, mais plus rapide à entraîner (car il nécéssite un seul entraînement et non 3) et tester et plus élégant qu'au chapitre précédent. Nous avons introduit une nouvelle paramétrisation de boîtes tournantes, un RoI-pooling tournant et un coût de régression robuste. En faisant cela, nous avons amélioré grandement les performances sur des bases publiques et nous avons démontré de très bons résultats sur des bases internes réelles et synthétiques (non présentées dans la thèse). Il serait toujours possible de continuer à améliorer notre détecteur en jouant sur : la prise en compte du contexte, les différents coûts, les architectures, etc. Mais le facteur limitant dans notre cas reste majoritairement lié au manque de données d'entraînement [3]. Ce qui serait préférable ce serait d'apprendre sur des bases synthétiques dont les tailles pourraient être illimitées et qui sont faciles à constituer, et d'avoir ensuite de bonnes performances sur des situations réelles (adaptation de domaine). Pour commencer à répondre à cette problématique, nous nous interrogeons dans le chapitre suivant sur l'adaptation de domaine en utilisant une des dernières techniques en vision par ordinateur : les GANs [75].

3. ainsi qu'à la puissance de calcul, mais c'est une autre question

class	F. R-CNN	FCN[170]	Ours
cars	77.69 ± 3.4	77.80 ± 3.3	**80.2 ± 3.3**
camp. cars	72.72 ± 5.6	—	**72.8 ± 4**
pickups	74.84 ± 2.77	—	**77.0 ± 4.2**
tractors	54.43 ± 7.00	—	**67.8 ± 12.1**
trucks	66.73 ± 9.97	—	**72.3 ± 7.6**
vans	69.93 ± 9.13	—	**74.1 ± 11.0**
boats	**66.22 ± 7.99**	—	65.3 ± 8.3
others	43.39 ± 11.45	—	**51.0 ± 8.8**
planes	**77.85 ± 34.45**	—	77.4 ± 32.7
mean	67.09 ± 13.6	—	**70.88 ± 13.3**

TABLE 4.2 – Comparaison de Faster R-CNN et Faster RER CNN sur VeDAI (10 folds)

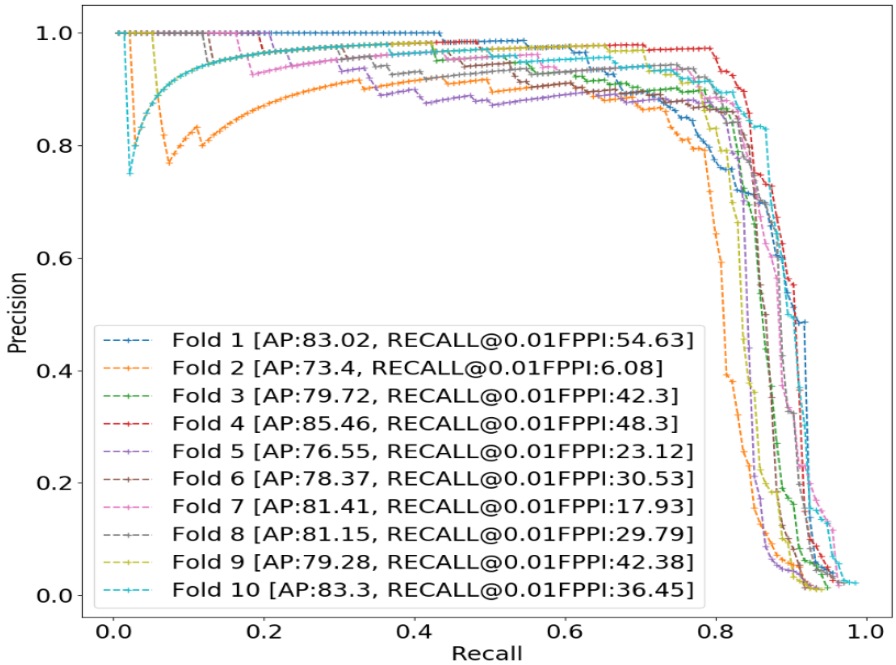


FIGURE 4.17 – Courbes précision-rappel du détecteur de voitures extrait de Faster RER-CNN

method	AP VOC@0.3	AP VEDAI
Faster R-CNN [196]	85.59	85.68
Ours	**87.14**	**87.32**

TABLE 4.3 – Comparaison Faster RCNN vs framework sur Munich3k

method	AP VOC@0.5	AP VEDAI
Faster R-CNN [196]	84.81	87.37
Ours	**88.39**	**88.53**

TABLE 4.4 – Comparaison Faster RCNN vs framework sur GoogleEarth

méthode	calcul de gradient et transformation des poids (s)	inférence(s)
Faster R-CNN [196]	0.362	0.158
Faster RER-CNN	0.602	0.365

TABLE 4.5 – Comparaisons des durées des différents frameworks pour une image 1024x1024 sur un Intel i7-7700K CPU@8*4.2GHz avec une GTX1080 (moyennées sur 100 runs)

classe	F. RER-CNN	Align F. RER-CNN
cars	80.2 ± 3.3	**81.2 ± 2.4**
camp. cars	72.8 ± 5.6	**73.2 ± 3.5**
pickups	77.0 ± 4.2	**78.56 ± 5.2**
tractors	**67.8 ± 12.1**	57.35 ± 9.34
trucks	72.3 ± 7.6	67.66 ± 8.4
vans	74.1 ± 11.0	67.90 ± 13.9
boats	65.3 ± 8.3	**66.06 ± 12.13**
others	51.0 ± 8.8	47.81 ± 11.17
planes	77.4 ± 32.7	73.04 ± 39.2
mean	70.88 ± 13.3	68.11 ± 15.7

TABLE 4.6 – Comparaison de Faster RER-CNN et de Align-Faster RER CNN sur VeDAI (10 folds)

méthode	AP	Rappel@0.01FPPI
DPM [189]	60.5 ± 4.2	13.4 ± 6.8
SVM+HOG31 [189]	55.4 ± 2.6	7.8 ± 5.5
SVM+LBP [189]	51.7 ± 5.2	5.5 ± 2.2
SVM+LTP [189]	60.4 ± 4.0	9.3 ± 3.7
SVM+HOG31+LBP [189]	61.3 ± 3.9	8.3 ± 5.2
SVM Fusion AED (HOG) [188]	69.6 ± 3.4	20.4 ± 6.2
Wide LeNet-5+HM (CV) [170]	77.80 ± 3.3	31.04 ± 11
Faster R-CNN (notre implémentation)	77.69 ± 3.4	20.85 ± 14
Faster RER-CNN	80.2 ± 3.3	33.2 ± 14
Faster RER-CNN Align	**81.2 ± 2.4**	**41.9 ± 14**

TABLE 4.7 – Comparaisons de l'existant avec Faster RER-CNN sur VeDAI sur la classe voiture

squelette	LeNet-5[21]	VGG-16[217]	ResNet-50^J[90]	ResNet-50^O[90]
AP Munich (PASCALVOC@0.3)	41.40	84.77	85.43	86.91
AP Munich (VeDAI)	41.63	85.01	85.57	86.97

TABLE 4.8 – Étude des différents squelettes pour Align-Faster RER-CNN sur Munich

Chapitre 5

Données synthétiques et modèles génératifs pour l'entraînement des détecteurs

Ce chapitre adresse une limitation essentielle des travaux précédents : celle de la disponibilité d'exemples pour l'entraînement des algorithmes. C'est particulièrement vrai dans les applications qui nous intéressent où collecter des données est difficile.

Le but de ce chapitre est de s'appuyer sur les avancées récentes des modèles génératifs CNNs pour la génération d'images pour générer des bases d'entraînement.

Nous commencerons ce chapitre par un aperçu des méthodes de création d'images artificielles, en l'orientant sur la création de base d'images d'entraînement. Ensuite, nous montrerons comment créer une première base hybride synthétique et réelle. Nous chercherons à rendre notre base hybride plus réaliste en utilisant les modèles génératifs au moyen du formalisme GAN. Des expériences seront conduites pour mesurer la capacité des images générées à entraîner des détecteurs d'objets. Nous dresserons finalement des conclusions et perspectives sur le travail réalisé.

5.1 Génération automatique d'images pour l'entraînement de détecteurs

La génération automatique d'image d'entraînement serait une avancée majeure dans le domaine de la vision par ordinateur. C'est encore plus vrai pour les CNNs : comme la capacité de ces réseaux est très grande, ils sont très limités par la quantité de données sur laquelle ils apprennent. La performance en généralisation est meilleure, nous l'avons vu, avec plus de données d'entraînement (voir l'exemple du sinus chapitre 1 ou même Wide LeNet-5 sur VeDAI dans le chapitre 2). Si le générateur est assez bon, un utilisateur pourrait donc synthétiser des millions d'images avec annotations gratuites et entraîner, par exemple, de vrais détecteurs robustes à l'aide des algorithmes vus dans le reste de la thèse.

Cependant, même si les moteurs de simulation physique utilisant du lancer de rayons ou de la rasterisation ont fait d'énormes progrès [6, 244, 34, 123, 22] – un exemple de ce que peuvent donner les simulateurs récents est présenté Figure 5.1) – les logiciels proposant assez de réalisme nécessitent ou bien des licences coûteuses [126], ou bien un savoir faire d'expert pour tirer pleinement partie de toutes leurs fonctionnalités (comme Blender [198] par exemple), et même dans ce cas, produisent toujours des images sensiblement différentes des images réelles (rendu visiblement synthétique même si réaliste). De plus certaines modalités comme l'Infra-Rouge ne sont pas développées dans la plupart des simulateurs grand public. La communauté du traitement d'image s'est donc depuis longtemps intéressée à la génération d'images à partir d'une collection ou d'un exemple en utilisant les propriétés statistiques de l'exemple (ou de la collection).

La génération de textures a été un des premiers domaines dans lequel il a été possible d'obtenir des images à peu près réalistes en utilisant simplement la répétitivité des motifs observables. Deux catégories d'approches sont alors possibles alors : celles dites structurelles fonctionnant mieux sur les textures complètement régulières et donc répétables, et celles probabilistes fonctionnant mieux avec les textures possédant une certaine stochasticité [82].

Dans les approches statistiques, nous pouvons citer [180], qui apprend à estimer

FIGURE 5.1 – Résultat de rendu temps réel d'images synthétiques présenté par NVIDIA au GDC 2018

la densité de probabilité de vecteurs multidimensionnels à valeurs discrètes (le pixel et son voisinage) : il clusterise d'abord l'ensemble d'apprentissage en M parties avec une variante plus ancienne de k-means. Puis la distribution de la texture est modélisée comme un mélange de gaussiennes centrées en ces clusters dont les paramètres sont estimés classiquement. Ce qui permet la génération récursive.

[93] utilise une pyramide laplacienne et de l'égalisation d'histogramme pour amener une image de bruit au fur et à mesure vers une texture cible en égalisant d'abord les détails grossiers puis tous les autres niveaux de la pyramide et cela de manière itérative.

[172] étend les précédents travaux en considérant des voisinages/zones d'influence plus grands en utilisant le formalisme des MRF.

Mais ce n'est vraiment qu'avec les travaux de Efros [53] que la génération de texture arrive à maturité et permet même de faire de l'inpainting sur des images complexes. Efros étend le travail de [180] en permettant la synthèse pixel par pixel à l'aide de MRF comme [172].

Quantité de travaux ont été publiés à la suite de ceux d'Efros. Nous décidons délibérément d'en ignorer la plupart pour nous focaliser sur les outils en rapport avec nos besoins et référons le lecteur intéressé au survey [242] pour faire le lien entre le présent et les travaux d'Efros de 1999.

Ce petit historique nous permet maintenant d'introduire nos outils de travail à savoir les GANs [75] de Goodfellow. Ils ont produit dans le domaine de la synthèse d'images des

bouleversements d'ampleur comparables à ceux de l'arrivée des CNNs dans la classification d'images. Les modèles génératifs, profitant des avancées en matière de CNNs, sont devenus très vite bien plus simples et plus puissants que la majorité de leurs prédécesseurs.

En plus de la génération de textures toujours plus complexes [119] les GANs arrivent aussi à produire des images complexes sans pour autant trop recopier les images observées ("mode collapse"). Ceci est vrai, que les vignettes soient des chambres d'hôtel ou bien même des classes d'Imagenet, avec cependant moins de facilité [182].

Depuis, une littérature importante s'est développée pour affiner ces techniques et les rendre plus stables. Nous ne citerons que les plus influents comme l'introduction du transport optimal pour modéliser la distance entre la distribution des images du générateur et celle réelle [77], les moindres carrés pour aligner la sortie du discriminateur sur la sortie voulue [159], la normalisation spectrale [163, 260] les GANs pour la super-résolution [125], etc. C'est donc tout naturellement que nous nous tournons vers ces techniques pour proposer un nouveau cadre pour entraîner nos détecteurs. Comme précédemment nous encourageons fortement le lecteur non familier avec cette littérature des modèles génératifs à base de CNNs à consulter l'annexe B avant de poursuivre.

Ce que nous voulons réaliser par le biais de la création d'images synthétiques est d'entraîner un détecteur sur un domaine A pour qu'il fonctionne sur un domaine B. C'est le problème de *l'adaptation de domaine*. Il y a peu d'articles traitant ce sujet spécifiquement *dans le contexte de la détection d'objets pour des CNNs* –[228, 223, 253] l'ont fait pour des descripteurs HOG – même si la littérature d'adaptation de domaine en classification est assez large [38]. Par exemple lorsque [233] entraîne Faster R-CNN sur COCO et veut le tester directement sur les voitures de KITTI [66] (la voiture est une des classes de COCO) il obtient 56.1% AP par rapport à 83.7% utilisant des images synthétiques plus proches du domaine d'arrivée.

La plupart des travaux adaptent les descripteurs appris dans un autre domaine simplement en fine-tunant les poids sur la tâche considérée. Par exemple depuis [71], tous les détecteurs état de l'art sont pré-entraînés sur ImageNet. C'est le cas même pour des BDD relativement larges comme COCO. Il n'y a pas de raison fondamentale qui

explique cela. Les objets du domaine source doivent être similaires en formes et en taille aux objets du domaine cible comme le montre [218]. Le travail de [101] montre comment transformer un bon classifieur entraîné sur une BDD large en un bon détecteur entraîné sur seulement quelques images en fine-tunant les premières couches d'un CNN et en adaptant le dernier descripteur (la dernière couche) en utilisant la proximité des classes. [99] a démontré que les meilleures capacités de transfert sont obtenues lorsque les premières couches de différents détecteurs état de-l'art entraînés sur des données synthétiques sont figées après entraînement sur données synthétiques.

Les articles allant plus loin que le simple fine-tuning sont rares. [183] aligne les descripteurs obtenus en entraînant dans différents domaines en utilisant l'analyse en composantes principales (PCA).[61] implémente un gradient jusqu'alors mal défini pour tromper un classifieur de domaine opérant sur les descripteurs. ADDA [234] adapte aussi les descripteurs dans une approche adversaire mais purement discriminative c'est-à-dire sans générateur. [27] utilise la théorie de la divergence H déjà mentionnée dans [61] et de l'entraînement adversaire pour combler l'écart entre les distributions réelle et synthétique. Tous les articles mentionnés adaptent les descripteurs. Récemment quelques-uns essayent d'adapter les domaines directement au niveau de l'image comme [114], qui utilise le CycleGAN [267] pour convertir directement les images entières dans le domaine cible. C'est l'article le plus proche de nos travaux. On peut aussi citer CycADA [102], qui combine l'adaptation des descripteurs et l'adaptation pixellique.

L'usage de données synthétiques pour entraîner les détecteurs est, lui, plus répandu. [176] fut, à notre connaissance, le premier à montrer que, même si les CNNs sont sensibles à la texture, des modèles fils de fer et CAO utilisés en conjonction avec des données réelles peuvent améliorer les performances d'un détecteur. Dans le même ordre d'idées [175] augmente PASCAL-VOC avec des modèles 3D CAO des objets trouvés dans PASCAL-VOC (avions, chevaux, plantes en pots, etc.), et utilise un moteur de rendu pour projeter les objets dans des lieux où ils ont des chances d'être trouvés statistiquement. En suivant cette direction générale de nombreux auteurs ont créé des données synthétiques pour plusieurs tâches et domaines, on peut citer i) les humains : [237] ii) les meubles : [160] . iii) le texte : [79] iv) les logos : [222]. [67] "copie colle" des rendus de CAO et les place sur

des surfaces normales en respectant la géométrie et [184], utilise Blender pour remplir un réfrigérateur. Plus tard [51], avec la même approche, mais sans utiliser de consistance globale, montre des promesses. De la même manière que [67], [51] utilise différents blendings pour éviter les artefacts de collage (plus de détails peuvent être trouvés dans [51]) ce que nous ferrons aussi. Plus récemment, [49] étend [51] en trouvant d'abord les régions dans lesquelles il est vraisemblable de trouver un objet. Une approche récente [233] trouve que la randomisation de domaine est importante pour obtenir de bonnes performances sur une base réelle. Virtual KITTI [59], une BDD qui a été construite pour être proche de KITTI (en termes d'aspects de textures et de statistiques de boîtes), n'est pas optimale pour transférer sur KITTI. L'auteur gagne au moins un point d'AP sur l'état de-l'art en construisant sa propre version de VirtualKITTI en introduisant plus d'aléatoire, que ce qui était initialement présent dans le VirtualKITTI originel. L'auteur introduit des textures aléatoires des fonds aléatoires, des angles de caméras aléatoires et des objets volants pour forcer le détecteur à se concentrer sur les véhicules. Ce travail motive aussi notre démarche.

5.2 Proposition d'une première méthode de génération d'images d'entraînement

Nous nous plaçons ici dans le cas où l'utilisateur dispose d'une base synthétique *d'objets*, potentiellement peu réaliste (ou simplement d'une modalité différente que celle souhaitée), *accompagnés de leur masque de segmentation sémantique*. C'est une hypothèse raisonnable, car, par exemple, tous les moteurs de rendus à partir d'un modèle CAO donnent la segmentation de l'objet par rapport au fond.

Nous nous plaçons également dans l'hypothèse dans laquelle l'utilisateur dispose aussi d'une collection des mêmes objets dans la modalité voulue, mais *sans forcément disposer des masques de segmentation*, ainsi que d'une collection de fonds. C'est le cas par exemple si l'utilisateur dispose d'une base de détections d'objets annotée par boîtes englobantes.

Le but des recherches présentées ici est de créer à partir de ces données faciles à

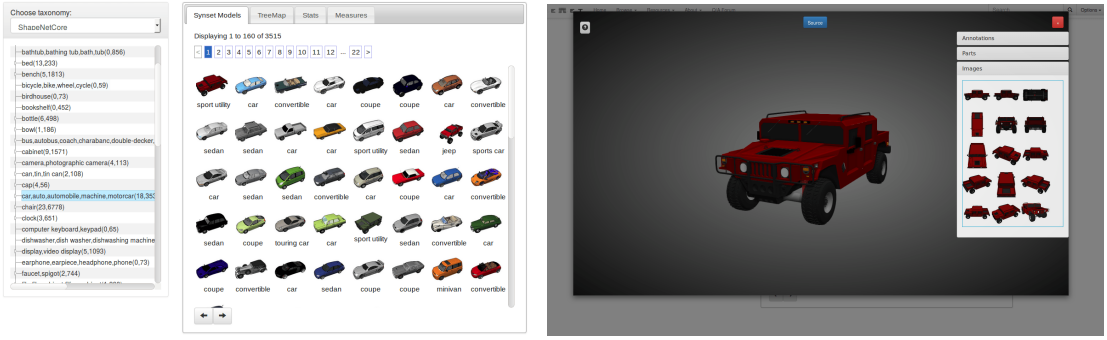


FIGURE 5.2 – Le site de ShapeNet et un exemple de voiture synthétique

obtenir une base d'entraînement dans la bonne modalité possédant, en grande majorité, en plus des boîtes englobantes, des masques de segmentation sémantique.

Dans le cas où cette opération serait possible, cela permettrait :

— de robustifier un détecteur entraîné sur trop peu d'exemples

— d'entraîner des systèmes de détection utilisant la segmentation sémantique pour améliorer la détection comme mask-RCNN [87] par exemple.

Dans la suite de ce chapitre, nous nous placerons dans notre cadre de détection de petits véhicules sur des images mal-résolues même si cette formulation est générale.

Les images synthétiques de voitures accompagnées de leur masque de segmentation sémantique seront obtenues depuis ShapeNet [23] qui dispose de vues du dessus cartoonesques (comprendre non-réalistes) d'une très grande variété de voitures, comme le montre la Figure 5.2.

5.2.1 Méthode naïve de génération d'images par copier/coller

La méthode présentée dans cette section n'a pas vocation à être réellement utilisée, mais permet d'introduire certains concepts clés. En particulier, elle fait abstraction de la différence de modalité en supposant que les images synthétiques sont assez proches des images réelles.

Cette méthode s'inspire de l'approche de détection d'instances proposée par [51] en "copiant-collant" les véhicules "cartoon" (c'est-à-dire ne suivant pas la réalité de la base de données réelles associée), en utilisant leur masque sur les images de la BDD utilisée pour l'entrainement (base Munich dans notre cas comme nous le verrons) en prenant

garde à ce qu'il n'y ait pas de recouvrement entre les objets. Selon [51], finalement, même si les véhicules collés ne sont pas à des endroits forcément réalistes seule la consistance locale au niveau du patch est importante. Dans les cas d'application qui nous intéressent, ce n'est pas une trop forte hypothèse, car nous voulons détecter des véhicules *pouvant être présents n'importe où*, même dans des contextes originaux. Cela nous empêche aussi de considérer les recherches de [50] se servant du contexte pour placer les objets créés. Nous pouvons alors obtenir une base non infinie théoriquement, mais de manière pratique déjà quasi infinie, car nous avons le choix : de l'orientation, de la taille ainsi que de l'emplacement où nous voulons coller nos véhicules, ce qui apporte une énorme variabilité. En cela, nous faisons écho aux très récentes recherches de [199], qui, compte tenu du fait que les masques sémantiques sont disponibles dans la BDD COCO, testent des détecteurs pré entraînés sur COCO en rajoutant des objets "copiés-collés" dans les images de validation. Ils s'aperçoivent de cas d'échecs impressionnants, ce que nous voulons éviter en apprenant sur une base plus grande possédant cette variabilité.

Bien sûr, une difficulté est que *les véhicules ne sont pas réalistes*, mais nous verrons cela dans un second temps. Pour éviter les artefacts de collage nous utilisons différentes sortes de blending comme proposé par [51] pour éviter que le détecteur ne se focalise sur des artefacts caractéristiques des véhicules générés. Soit $\mathcal{R}$ la BDD réelle et S la collection de vignettes synthétiques *avec masques*. Nous allons maintenant présenter l'algorithme utilisé. Nous supposons qu'on dispose d'une fonction `find_box` qui permet à partir d'un masque de trouver la plus petite boîte orientée le contenant (en pratique elle est implémentée dans OpenCV). Cette fonction renvoie les sorties suivantes : w,h et θ respectivement le plus grand côté, le plus petit ainsi que l'angle de la boîte par rapport à l'horizontale. Nous disposons aussi d'une fonction $\texttt{sim}(i, r, \theta)$ qui effectue une homothétie de l'image i de rapport r suivi d'une rotation d'angle θ de l'image. Enfin $W_{\mathcal{R}}$ est la liste des plus grands côtés de toutes les boîtes orientées vérité terrain de $\mathcal{R}$, N_S est le nombre d'exemples synthétiques voulus *par image*, et $boxes_r$ représente la liste des boîtes orientées vérité terrain dans l'image $r \in \mathcal{R}$, qui est aussi contenue dans un fichier texte $annotations_r$. La dernière fonction utilisée *Blend* sera discutée plus avant en fin de section. L'algorithme 3 résume les opérations à réaliser pour créer la base.

Algorithm 3 Construire base hybride $\mathcal{R} + N_S * S$

Require: $\mathcal{R} \vee S \vee N_S$
  $\hat{m}_W \leftarrow MEAN(W_{\mathcal{R}})$
  $\hat{\sigma}_W \leftarrow STD(W_{\mathcal{R}})$
  for $r \in \mathcal{R}$ **do**
    $mask_r = 0$
    $c \leftarrow 0$
    while $c < N_S$ **do**
      $choisir \quad s_c \quad dans \quad S \quad avec \quad remplacement$
      $(w_c, h_c, \theta_c) \leftarrow$`find_box`$(mask(s_c))$
      $w_t \leftarrow U_{[\hat{m}_W - \sqrt{3}\hat{\sigma}_W, \hat{m}_W + \sqrt{3}\hat{\sigma}_W]}$
      $\theta_t \leftarrow U_{[0,180]}$
      $s'_c \leftarrow$`sim`$(s_c, \frac{w_t}{w_c}, \theta_t)$
      $mask(s'_c) \leftarrow$`sim`$(mask(s_c), \frac{w_t}{w_c}, \theta_t)$
      $box_c \leftarrow$`find_box`$(mask(s'_c))$
      `find_position`$\leftarrow False$
      while ¬`find_position` **do**
        $(x_t, y_t) \leftarrow U^2_{[0,taille_{image}]}$
        $box_c \leftarrow box_c + (x_t, y_t)$
        if $(\cup boxes_r) \cap box_c = \emptyset$ **then**
          `find_position`$\leftarrow True$
        else
          $continue$
        end if
      end while
      $r \leftarrow Blend(r, s'_c, (x_t, y_t), MODE)$
      $UPDATE(annotations_r, box_c)$
      $mask_r \leftarrow mask_r + TRANSLATION(mask(s'(c)), (x_t, y_t))$
      $c \leftarrow c + 1$
    end while
  end for

En pratique dans tout le reste de la thèse $\mathcal{R}$ sera Munich et la collection S sera issue ou bien de ShapeNet ou bien de résultats de modèles génératifs.

Les différents blendings utilisés dans la fonction *Blend* sont :

Pas de blending. Le masque du véhicule déformé est placé dans l'image et l'image résultante est formée des pixels de l'image quand le masque est nul, ou des pixels de la voiture quand le masque est à 1.

Blending moyen, médian ou gaussien. Une fois la stratégie sans blending utilisée un filtre 3x3 est appliqué à l'image, il n'est activé que si il intersecte le masque par au moins un pixel et est désactivé s'il est entièrement contenu dans le masque, ainsi seul le bord du véhicule est affecté. Les différents filtres de convolution utilisés sont :

— le filtre moyen :

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \tag{5.1}$$

— le filtre gaussien :

$$\begin{pmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{pmatrix} \tag{5.2}$$

— le "filtre" médian (c'est un abus de langage, car le filtre dépend de la position) avec un 1 à l'emplacement de la médiane des pixels sur le filtre et 0 partout ailleurs

Évidemment cela donne des idées pour apprendre un blending par deep-learning mais nous y reviendrons.

Blending Laplacien pyramidal [19]. Une *pyramide laplacienne* de notre image de fond est construite, puis notre voiture est collée sans blending. C'est une pyramide formée à partir de l' image originale en appliquant successivement un filtre Gaussien et un sous-échantillonnage d'un facteur 2, les résidus, permettant de retrouver les différents niveaux de la pyramide à partir de la version floutée par le filtre gaussien, sont stockés et

permettent de reconstruire l'image à partir de la pyramide *sans perte*. Nous construisons aussi différents masques obtenus en sous-échantillonnant le masque initial du véhicule successivement. À chaque niveau de la pyramide, en commençant par le plus bas, nous fusionnons les deux niveaux correspondants des deux images en utilisant la version de même résolution du masque. Ainsi le blending est moins brutal et nous observons une belle transition entre les deux images. Il est cependant nécessaire de borner l'image obtenue, car les valeurs obtenues sortent souvent des bornes du codage en entiers sur 8 bits.

Le blending de Poisson [177]. Il s'agit de raffiner le blending précédent en utilisant une résolution numérique directe. Nous considérons l'image de destination ainsi qu'un découpage grossier de notre voiture *sur un fond*. Nous pourrons par exemple coller la voiture sur un fond gris et dilater le masque en utilisant des outils de morphologie mathématique. L'image de destination impose des conditions aux bords du masque (en termes de couleurs), la voiture collée *avec un peu de fond* donne un champ de gradient orientant. Et la résolution de l'équation de Poisson discrète permet de remplir la zone de la voiture collée avec un peu de fond en prolongeant les couleurs aux bords tout en suivant le champ de vecteur. Le masque doit absolument être dilaté, car sinon le gradient aux bords de notre véhicule est considéré comme nul et il disparaît dans l'image de destination. Il est donc important de trouver un fond neutre pour avoir du gradient aux bords de notre véhicule, mais pas trop sinon le respect du champ de gradient introduit forcera l'image remplie à être très différente des conditions aux limites ce qui fera ressortir encore plus notre voiture. Nous utilisons l'implémentation d'openCV `seamless_clone`.

Différents exemples d'une même voiture générés avec tous les types de blending sont représentés Figure 5.3.

Nous donnons aussi une version moins zoomée et plus représentative de cette nouvelle base hybride synthétique et réelle Figure 5.4

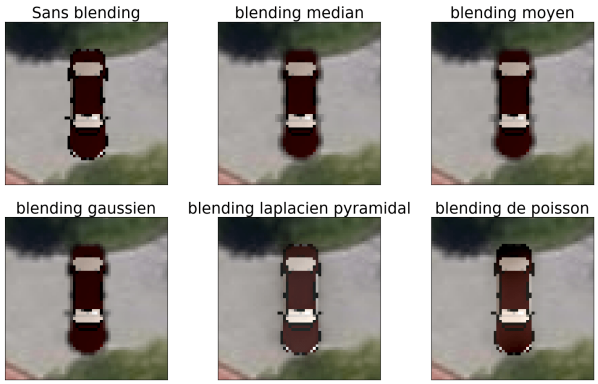


FIGURE 5.3 – Voiture issue de Shapenet collée sur un fond de la BDD Munich avec différentes stratégies de blending

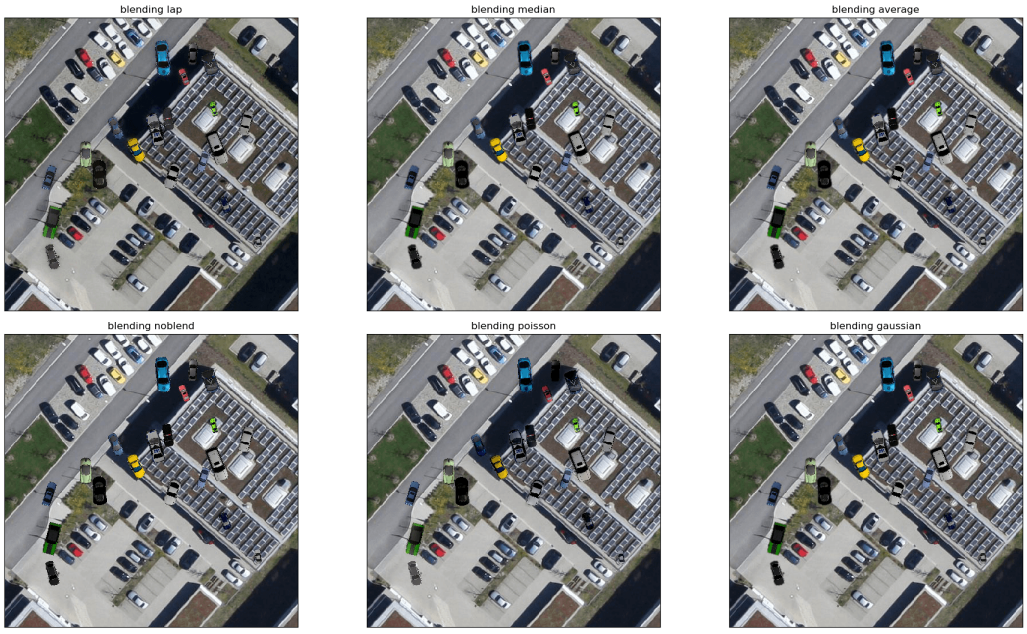


FIGURE 5.4 – Base de Munich hybride obtenue avec la baseline Shapenet

5.2.2 Méthode m-CycleGAN

La méthode m-CycleGAN que nous proposons ici s'inspire de la méthode naïve précédente, mais les images de ShapeNet sont remplacées par des images générées automatiquement de manière à effectuer une adaptation de domaine : les voitures de ShapeNet n'étant pas réalistes, nous décidons de remplacer la collection S d'images synthétiques utilisées en utilisant des modèles génératifs que nous présentons.

La méthode proposée repose sur CycleGAN [267] qui permet de traduire des images d'un domaine à un autre. De manière plus précise, elle permet, à partir de deux collections d'images A et B de différentes modalités (par exemple réel et synthétique pour lier cette section au paragraphe précédent), de réaliser un traducteur de l'un à l'autre en utilisant des réseaux Deep de type U-Net [197] ou ResNet[90]. De plus, la méthode ne nécessite pas de disposer de paires d'images pour l'entraînement. Cela est rendu possible grâce à la consistance de cycle qui permet de créer une supervision : en chaînant les deux traducteurs, on devrait retrouver l'image de départ dans un sens ou dans l'autre. Si nous posons :

$$F : \begin{pmatrix} A \to B \\ a \mapsto F(a) \end{pmatrix} \quad G : \begin{pmatrix} B \to A \\ b \mapsto G(b) \end{pmatrix}$$

avec F et G paramétrisés comme habituellement par des réseaux de neurones convolutionels. Le coût de consistance de Cycle qu'il faut minimiser s'écrit :

$$C_{cycle}(W_F, W_G) = \mathbb{E}_{b\sim p_B(b)}\|F(G(b)) - b\| + \mathbb{E}_{a\sim p_A(a)}\|G(F(a)) - a\| \tag{5.3}$$

Où $\|\|$ est une norme quelconque, celle choisie dans l'article est la norme L1. Bien sûr F et G peuvent apprendre l'identité, ce qui n'est évidemment pas souhaitable. Pour forcer F et G à apprendre une transformation passant bien d'un domaine à l'autre on utilise le formalisme GAN évoqué précédemment en introduisant deux discriminateurs D_A et D_B.

$$C_{GAN}(W_F, W_G) = \mathbb{E}_{b\sim p_B(b)}(D_A(G(b)) - 1)^2 + \mathbb{E}_{a\sim p_A(a)}(D_B(F(a)) - 1)^2 \tag{5.4}$$

Nous avons utilisé en pratique LS-GAN [159] qui possède un coût plus intuitif et plus stable que dans la formulation GAN originale. Les discriminateurs D_A et D_B sont entraînés de manière alternative à minimiser :

$$\begin{aligned} C_{GAN}(W_{D_A}, W_{D_B}) =& \mathbb{E}_{b\sim p_B(b)}(D_A(G(b)))^2 + \mathbb{E}_{a\sim p_A(a)}(D_B(G(a)))^2 \\ &+ \mathbb{E}_{b\sim p_B(b)}(D_B(b)-1)^2 + \mathbb{E}_{a\sim p_A(a)}(D_A(a)-1)^2 \end{aligned} \tag{5.5}$$

Cette approche ne peut malheureusement pas directement être appliquée sur nos images, car les véhicules de la BDD Munich possèdent du fond et ceux de ShapeNet n'en possèdent pas. De plus *nous voulons conserver a tout prix le masque sémantique* : comme nous voulons les coller après il n'est pas possible a-priori de conserver du fond (cette observation sera nuancée dans la section suivante).

Nous utilisons donc une variante de ce qui est proposé dans [15] que nous appelons m-CycleGAN. Nous commençons par extraire des masques grossiers sur les images réelles à l'aide de GrabCut [200] initialisée en utilisant les boîtes tournantes vérités terrains. Nous désactivons ensuite le coût de reconstruction sur les zones non présentes dans le masque donnant la liberté aux deux traducteurs de créer n'importe quel fond en dehors du masque. Nous faisons de même dans les discriminateurs, qui sont complètement convolutionnels (PatchGAN). Le discriminateur ne doit pas s'appuyer sur le fond pour dire si l'exemple est faux ou non :

$$C_{m-CycleGAN}(w) = GAN((G(s)\circ m_s, \mathbf{R})) + GAN((F(t)\circ m_t, \mathbf{S}))+ \tag{5.6}$$

$$\lambda\left(||(F(G(s))-s)\circ m_s||_1 + ||(G(F(t))-t)\circ m_t||_1\right) \tag{5.7}$$

Où S et T désignent les domaines Synthétiques et Réels. Nous obtenons un traducteur quasi parfait, qui nous permet de créer une image de modalité réelle à partir de chaque membre de notre collection synthétique. Nous vérifions la pertinence de notre approche en considérant la baseline où le jeu de données masqué est utilisé directement par GrabCut dans un CycleGAN traditionnel. L'avantage de notre méthode Figure 5.5 est incontestable. Nous avons donc réussi, pour chaque voiture de ShapeNet, à obtenir une voiture réaliste *avec son masque* (nous reviendrons section 5.3.2 sur la signification

FIGURE 5.5 – A gauche des paires d'exemples créées avec CycleGAN en masquant les images réelles directement avec GrabCut, à droite les paires créées en utilisant m-CycleGAN

du mot réaliste). Nous pouvons dès à présent utiliser ces véhicules comme nous le faisions avec les véhicules de ShapeNet à la section précédente. Nous avons donc créé aussi un dataset apparié R-S (Réel-Synthétique) dont des exemples sont présentés Figure 5.6.

Le processus est conçu pour que les véhicules générés respectent le masque de l'objet synthétique. En réalité en zoomant sur le bord du masque des véhicules générés, il y a parfois un décalage d'un ou deux pixels entre le masque théorique et le masque réel. Cela introduit des pixels parasites. Les blendings gaussien, moyen ou médian sur les bords viennent tempérer cet effet.

Compte tenu des limitations dans le images générées, nous avons choisi de ne pas chercher à entraîner des détecteurs à partir de ces images, mais plutôt de proposer une méthode encore meilleure.

5.2.3 Méthode TriCycleGAN

La méthode de génération appelée TriCycleGAN, que nous proposons ici, suit les mêmes hypothèses que les hypothèses précédemment évoquées.

Elle s'appuie en particulier sur le BiCycleGAN [268], qui n'a pas grand-chose à voir avec CycleGAN comme ne l'indique pas son nom. Cette architecture permet *à partir d'une base de données de paires* d'entraîner un modèle génératif. La formulation

FIGURE 5.6 – Extrait de la BDD *de paires* créée par m-CycleGAN montrant la robustesse du système aux voitures non standards de ShapeNet

exacte sera décrite plus loin plus en détail, mais pour simplifier il s'agit d'une architecture encodeur-décodeur avec un a-priori gaussien sur le code z à la VAE[1], qui permet d'échantillonner de nouvelles images en générant de nouveaux z à partir d'une loi normale.

Nous venons juste de créer notre base de paires avec CycleGAN. Nous sommes donc parfaitement dans le cadre d'utilisation de BiCycleGAN.

Pour comprendre les intuitions derrière BiCycleGAN il faut revenir à pix2pix [116]. Isola et al. proposent d'apprendre une traduction d'un domaine vers un autre *de manière supervisée*. Le mécanisme fonctionne comme CycleGAN, mais la consistance de cycle est remplacée par une supervision directe qui est disponible dans les bases étudiées par [116]. Il est très facile en théorie de remplacer pix2pix par un modèle génératif. Il suffit de conditionner non seulement par rapport à l'image *mais aussi par rapport à une variable aléatoire qui peut être introduite n'importe où dans le réseau [268].* Seulement en entraînant de cette manière, le générateur apprend à ne pas utiliser le bruit. Effectivement, il n'y aucun terme dans la fonction de coût pour le forcer à utiliser z. Pour vraiment encourager le générateur à être stochastique, les auteurs proposent 2 termes :

cVAE-GAN Pour forcer à ce que z soit utile, on utilise un auto-encodeur pour que le modèle soit capable de régénérer l'image à partir du code obtenu. Une image du domaine B notée b est transformée par le goulot d'étranglement d'un auto-

1. Nous renvoyons le lecteur sur l'Annexe B pour plus d'information sur les VAEs

encodeur en un code z_b. Et ce code est ensuite utilisé *avec l'image A* pour essayer de reconstruire b (similairement à pix2pix+bruit sauf qu'ici z est obtenu à partir de b et donc lui est lié) :

$$\mathcal{L}_1^{VAE} = \mathbb{E}_{(a,b)\sim p_{A,B}(a,b),z\sim E(b)}\|b - G(a,z)\| \tag{5.8}$$

Il ne peut donc pas reconstruire b si z_B ne contient pas les informations nécessaires sur b. Parallèlement, en utilisant un code simulé à partir d'une gaussienne d'écart-type et moyenne encodé par b et l'image a, on doit produire une image dans le domaine de B on mesure cela avec le coût classique min-max (min pour le générateur, max pour le discriminateur) GAN :

$$\mathcal{L}_{GAN}^{VAE} = \mathbb{E}_{b\sim p_B(b)}(D(b)^2) + \mathbb{E}_{z\sim E(b),A\sim p_A(a)}((1 - D(G(z,a)))^2) \tag{5.9}$$

(L'équation est simplifiée, car en pratique on conditionne le discriminateur par le a utilisé pour générer l'image pour gagner quelques points de performance (voir [116])) Pour forcer le code à suivre une gaussienne on utilise :

$$\mathcal{L}_{KL}(E) = \mathbb{E}_{b\sim p_B(b)}(D_{KL}(E(b)||\mathcal{N}(0,I)) \tag{5.10}$$

En enlevant la divergence KL et en échantillonnant z à partir d'une gaussienne (non encodée) on retrouve exactement pix2pix+bruit. Normalement ce formalisme se suffit à lui même, mais ce n'est pas suffisant selon les auteurs, ils introduisent donc d'autres coûts.

cLR-GAN Nous échantillonnons un code z aléatoirement à partir d'une gaussienne donc correspondant à un B hypothétique et on essaye de le reconstruire à partir de l'image générée par A et ce code.

$$\mathcal{L}_1^{latente} = \mathbb{E}_{z\sim p(z),a\sim p_A(a)}\|z - G(a,z)\| \tag{5.11}$$

Nous ajoutons aussi la même équation que dans $\mathcal{L}_{GAN}^{VAE}$, mais le z est généré à

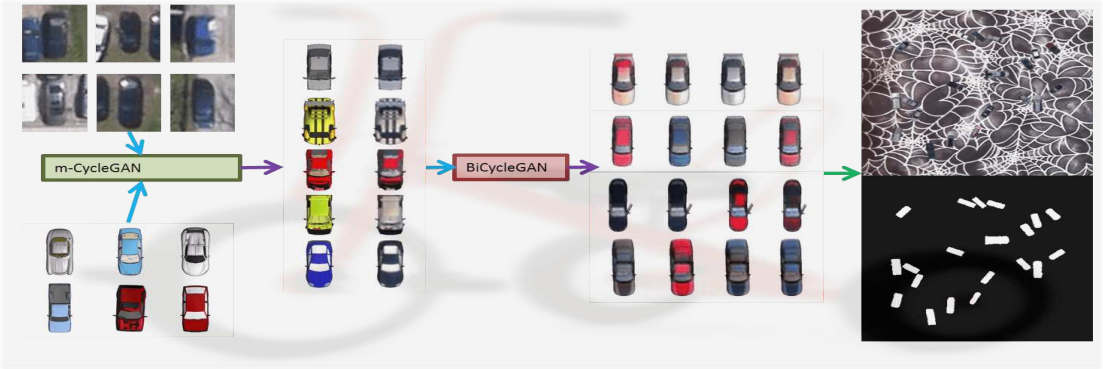


FIGURE 5.7 – TriCycleGAN en action : une BDD de paires est d'abord créée grâce à m-CycleGAN, puis on entraine BiCycleGAN sur ces paires. Les exemples générés par BicyleGAN sont ensuite warpés pour être "copiés-collés" avec différents blendings sur des images de fonds

partir d'une gaussienne et non encodé. Nous l'appellons $\mathcal{L}_{GAN}$.

En combinant les deux coûts nous obtenons le BiCycleGAN :

$$\mathcal{L}_{bicycleGAN} = \mathcal{L}_{GAN}^{VAE} + \mathcal{L}_{KL}(E) + \mathcal{L}_{1}^{VAE} + \mathcal{L}_{1}^{latente} + \mathcal{L}_{GAN} \tag{5.12}$$

En entraînant BiCycleGAN sur les paires créées par CycleGAN, nous obtenons pour chaque voiture de ShapeNet une infinité de voitures réalistes possédant le même masque.

La méthode TriCycleGAN repose donc sur en enchaînement de deux opérations :

— la génération de S par la chaîne CycleGAN-BiCycleGAN
— le "copier-coller", explicité par le pseudo-code de début de section présenté Figure 5.7.

En pratique, la diversité des exemples du modèle génératif reste limitée et, car nous disposons d'environ 5 échantillons vraiment différents. Nous utilisons ces échantillons comme précédemment.

Nous présentons un échantillon d'images générées à partir de Munich avec le TriCycleGAN Figure 5.8.

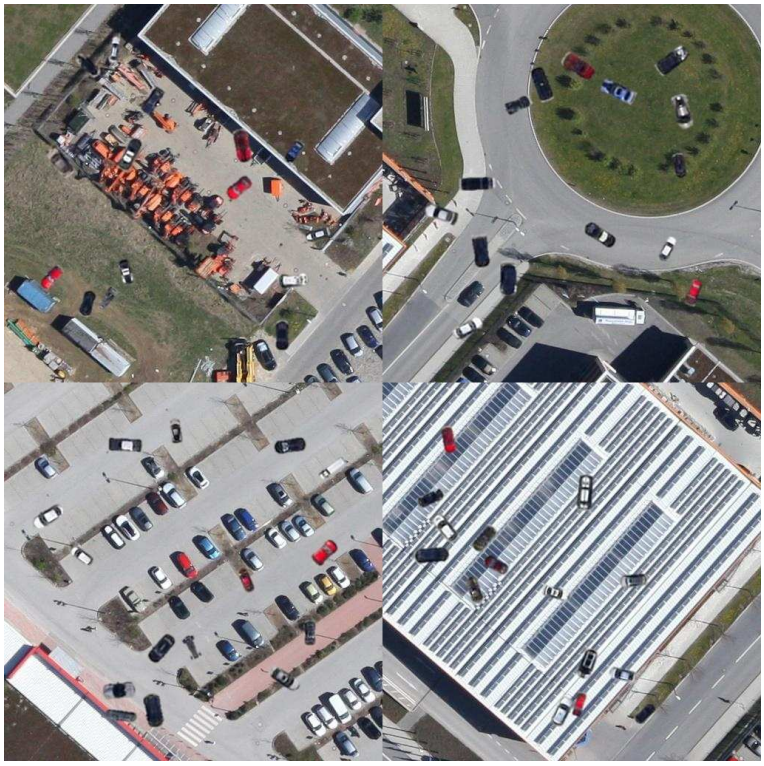


FIGURE 5.8 – TricycleGAN pour la BDD Munich hybride : 4 exemples d'images avec différents blendings. Le lecteur est encouragé à rechercher les 15 voitures synthétiques qui se cachent dans chaque image.

Apprentissage \ Test	Munich val	Munich+10S val	DTD+10S val
Munich	**87.4**	89.0	86.6
Munich+10S	73.6	89.6	90.7
DTD+10S	40.29	79.3	**90.9**
Munich+5S	82.2	**90.0**	90.2

TABLE 5.1 – Différentes AP de transfert sur la classe voiture (@0.5 IoU)

5.3 Évaluation expérimentale

Nous évaluons ici la capacité à entraîner un détecteur d'objet au moyen de la méthode TriCycleGAN. La base d'imagerie aérienne utilisée pour les expérimentations est la base Munich [141] déjà vue précédemment.

Nous essayons d'augmenter la base Munich en utilisant les voitures ShapeNet dans notre TriCycleGAN, afin de créer de meilleurs détecteurs. Le détecteur considéré dans cette section est le détecteur Faster R-CNN [196], qui est préféré au détecteur du chapitre précédent car plus rapide.

5.3.1 Performance des détecteurs entraînés sur les images générées

Nous entraînons ce détecteur Faster R-CNN sur différentes BDDs dont chacune comporte une partie apprentissage une partie validation et une partie test :

— la BDD Munich originale (Munich),

— la BDD Munich originale auquel nous rajoutons respectivement 5 et 10 voitures synthétiques par image ($\mathcal{R} = Munich$ et $N_S = 5$ ou 10) grâce à notre TriCycleGAN (Munich+5S, Munich+10S respectivement),

— Une BDD composée de fonds texturés issus de DTD [32] sur lequel sont collées 10 voitures synthétiques par image (comme les fonds sont vides au départ, les masques sémantiques sont parfaits. Nous pourrions donc entraîner un FCN [211] ou un Mask R-CNN [87] directement). Nous le nommons DTD+10S

Nous testons ensuite les capacités de transfert des uns vers les autres et présentons les résultats dans la Table 5.1.

Nous pouvons observer plusieurs choses sur cette table. Notre but final étant

d'améliorer les performances sur la BDD Munich, c'est un échec pour l'instant : il est toujours préférable d'entraîner sur les images complètement réelles pour tester sur Munich ainsi TriCycleGAN n'est pas complètement au point. Cependant on voit qu'en entraînant sur une BDD complètement synthétique de textures et de véhicules nous obtenons quand même 40.3 d'AP sur Munich, ce qui n'est pas rien compte tenu de sa difficulté. De plus, l'apprentissage sur images réelles se transfère relativement bien sur images complètement synthétiques avec quasiment 86.6 d'AP sur DTD+10S ce qui montre qu'un détecteur est trompé par nos exemples synthétiques (mais moins qu'un apprentissage hybride donnant 90.7 d'AP).

5.3.2 Mesure du réalisme des images produites

En partant de l'hypothèse que les mauvais résultats en détection viennent du faible réalisme des images (hypothèse de travail la plus simple mais pas forcément validée) il s'agit de voir à partir de quelle étape les résultats commencent à se dégrader, il faut donc avoir accès à un résultat quantitatif sur le réalisme des images produites. Il se trouve que c'est un sujet ouvert, mais il existe nombre de réponses partielles pour estimer si un échantillon synthétique est réaliste :

— le jugement humain,

— entraîner un classifieur sur les images réelles et mesurer sa confiance sur les images synthétiques : si elles sont de bonne qualité, le classifieur devrait être trompé

— le SSIM [241] (métrique de différence perceptuelle entre deux images basée sur des moyennes et écarts-types locaux) entre les images réelles et les images produites.

Et des métriques pouvant juger de la pertinence d'une *distribution* d'images synthétiques comme :

— le score Inception [202], qui est une extension du point précédent. Un classifieur est entraîné sur ImageNet pour obtenir la conditionnelle $p(y|x = G(z))$: elle doit être d'entropie faible i.e. le modèle doit être confiant qu'il y ait un objet (comme précédemment). Nous rajoutons aussi que le modèle doit produire des échantillons divers,

— le FID [97], un score mesurant la compatibilité des premiers moments des distri-

butions obtenues dans l'espace des descripteurs des deux ensembles d'images,

— le KID [12], similaire au FID, mais permettant grâce à l'usage de noyaux de se passer de l'hypothèse (non vérifiée), que les distributions soient gaussiennes dans le FID [97],

— la différence perceptuelle introduit par Gatys [64] aussi appelée "feature matching" où l'on regarde la différence entre les descripteurs à différentes couches d'un réseau pré entraîné obtenus pour les images réelles et pour les images synthétiques (VGG est dans ce cas curieusement bien meilleur que tous les autres réseaux plus récents).

Nous avons choisi le FID qui semble être le plus adapté. La raison principale de ce choix est que cette mètrique est calculée au niveau des descripteurs, ce qui nous intéresse le plus pour entraîner un Faster R-CNN. À la limite, que les images aient l'air réalistes est même moins important que le fait que leurs descripteurs soient proches de ceux des images réelles (même si les deux sont liés c'est ce qui fait la puissance des réseaux Deep). De plus, en appliquant le principe du rasoir d'Occam, le score Inception nous semblait d'interprétation moins directe, de même que le KID, même si nous savons que les hypothèses de calcul du FID sont fausses. Plus formellement, nous définissons la distance de Fréchet entre deux distributions comme :

$$W_2(\mathcal{X}, \mathcal{Y}) = \inf \mathbb{E}(\|X - Y\|_2^2)^{\frac{1}{2}} \tag{5.13}$$

Où les vecteurs X et Y sont échantillonnés à partir des distributions respectives. Si les distributions sont gaussiennes $\mathcal{N}(m_1, \Sigma_1)$ et $\mathcal{N}(m_2, \Sigma_2)$ nous avons alors la forme analytique suivante (le lecteur intéressé trouvera la dérivation dans [72]) :

$$d^2 = \|m_1 - m_2\|_2^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}). \tag{5.14}$$

Les distributions considérées pour le FID seront donc les distributions de descripteurs issus de Inceptionv3 [227] entraîné jusqu'à convergence sur ImageNet et nous les considérerons comme gaussiennes. Il s'agit donc juste d'estimer les moyennes et les ma-

Blending	ShapeNet	TriCycleGAN
–	155.98	**102.61**
Moyen	116.86	**82.67**
Médian	116.86	**82.63**
Gaussien	120.84	**84.44**
Laplacien Pyramidal	151.07	**99.83**
Poisson	145.42	**100.09**

TABLE 5.2 – Comparaisons des FIDs obtenus pour les mêmes véhicules (par colonne) et les mêmes fonds dans tout le tableau pour différents blendings

trices de covariance des descripteurs issus des deux distributions :

$$FID(R,S)^2 = \|\mathbb{E}_{x\sim R}(\phi(x)) - \mathbb{E}_{s\sim S}(\phi(s))\|_2^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}). \quad (5.15)$$

Bien sûr, le FID théorique entre deux ensembles générés par la même distribution doit être théoriquement nul. Maintenant, compte tenu des hypothèses et de la finitude des échantillons, en pratique, le FID entre différents ensembles finis de la même distribution est loin d'être nul. Nous le calculons pour les vignettes de voiture (rectifiées en angle pour faciliter le travail du réseau Inception qui n'est pas invariant à la rotation) de la BDD Munich entre elles (nous les séparons en 10 folds et faisons la moyenne) :

$$FID(R,R) = 37.74 \pm 0.18 \quad (5.16)$$

Ce sera donc notre borne inférieure théorique. Le FID de la baseline (les voitures "cartoon" collées sans blending sur les fonds de Munich , voir Figure 5.3) donne notre borne supérieure : 168.59.

Ainsi si nous obtenons un FID de l'ordre de 40-50 ; nous considérerons que les vignettes obtenues sont réalistes, et si le FID obtenu est supérieur à 170 alors, la chaîne est considérée comme un échec et nous ne perdons pas de temps à tester l'apprentissage d'un détecteur.

Nous représentons les FID obtenus pour différentes stratégies de blending pour les images de ShapeNet ainsi que celles de TriCycleGAN Table 5.2.

Nos exemples générés par notre TriCycleGAN collés sans blending sur des vignettes de fonds donnent 102.61. Nous nous sommes donc donc rapprochés de la borne théorique,

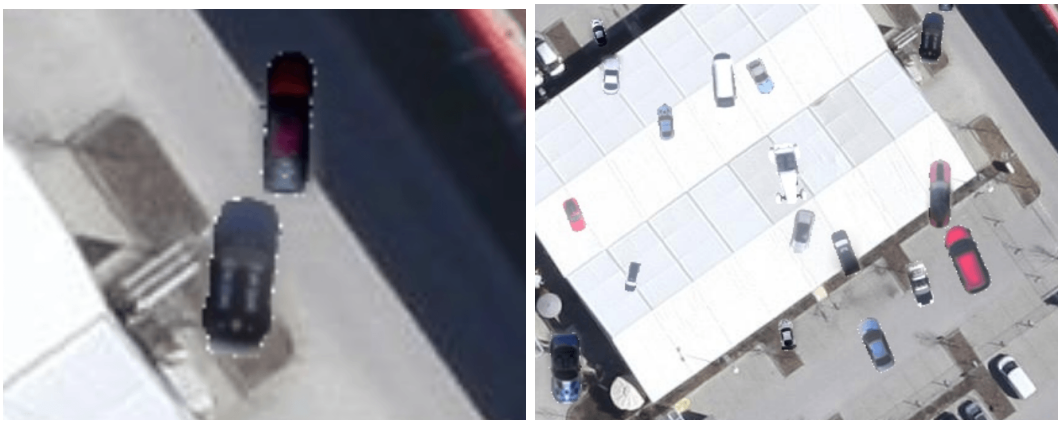


FIGURE 5.9 – Interêts et désavantages du blending de Poisson : à gauche le véhicule rajouté à l'interface ombre et lumière s'intègre bien dans son environnement ; A droite les véhicules sur le toit blanc disparaissent dans celui-ci

mais il existe encore de fortes différences de statistiques probablement liées aux artefacts de blending. En utilisant du blending médian ou moyen nous arrivons à baisser le FID des images produites à 83 environ, ce qui confirme l'hypothèse que le blending est important. Les blending plus compliqués semblent à première vue moins pertinents *seuls*, car il existe toujours des pixels blancs sur les contours des objets. Mais ils ont leur importance comme le montre la Figure 5.9. Il serait donc possible de combiner les deux types de blending en commençant par un blending de Poisson suivi d'un blending médian. Il serait aussi possible en pré entraînant un classifieur de refuser un blending si celui-ci est mauvais au sens du classifieur (ou bien au sens du FID simplement) ou bien même de choisir les blendings avec une approche par apprentissage par renforcement similaire à celle utilisée pour choisir les stratégies de data-augmentation dans Auto-Augment [39]. Le tableau présenté ici est une des premières justifications expérimentales sérieuses des mérites comparés des différentes stratégies de blending pour les détecteurs. À titre de comparaison le CycleGAN naturel entraîné sur des images *avec du fond* en collant les images de ShapeNet sur des fonds Munich avec un blending médian est meilleur *il donne 64.26 de FID*, mais il pose deux problèmes :

— il ne respecte pas le masque du véhicule, initial

— il va créer des fonds parasites en plus du véhicule.

Nous pourrions cependant, à l'aide du blending de Poisson, essayer de masquer les fonds créés en utilisant un masque grossier.

5.4 Limites de la méthode TriCycleGAN et axes d'améliorations

Cette section comporte une analyse des limitations des résultats précédents et tente, plus généralement, de dresser des pistes pour améliorer ces résultats.

Fonds difficiles pour la BDD Munich. Ajouter d'autres voitures similaires à celles présentes dans la base est important en général, mais les voitures ajoutées ne sont peut être pas assez différentes de celles déjà présentes, en particulier quant à leur intégration sur des fonds riches. Sa principale force dans ce cas viendrait du fait qu'il est maintenant possible, sans budget supplémentaire, de coller nos voitures dans des fonds difficiles texturés *non présents dans la BDD Munich*. DTD n'est cependant pas vraiment représentative de nos conditions de travail. Comme beaucoup de faux positifs sont présents dans les zones de forêt par exemple il serait facile de collecter les premières images de Google Images pour la requête "aerial imagery forest", de rajouter des voitures dans ces images et de les utiliser dans l'entraînement (le code originel de Faster R-CNN n'accepte pas les images contenant uniquement des fonds).

Régime "few-shots". Les voitures générées avec TriCycleGAN semblent trop réelles pour ne pas apporter de bénéfices sur l'apprentissage. Seulement il y a déjà beaucoup de voitures dans la BDD de Munich et donc l'influence des exemples ajoutés est minime. Pour se mettre dans le régime "low-shots" où l'on dispose de vraiment peu d'exemples vrais à partir de Munich il faudrait enlever la majorité des vérités terrains à l'aide d'inpainting en utilisant soit PhotoShop soit l'inpainting de [147], qui permettrait d'effacer toutes les voitures en masquant les vérités terrains puis en remplissant leur masque à l'aide du contexte. Nous pourrions alors vraiment juger de l'apport des voitures synthétiques. Une autre manière de se mettre artificiellement dans le même régime serait lors de l'entraînement de Faster R-CNN [196] de n'utiliser aucune région intersectant les vérités terrains. Ainsi cela serait comme si elles étaient invisibles pour Faster R-CNN.

Apport d'un TriCycleGAN parfait. Une autre expérience qui serait intéressante serait de segmenter parfaitement les vérités terrains existantes (mais comment le faire ?) et d'en coller des copies à différents endroits et différents angles *pour avoir une borne supérieure sur les performances obtenues avec blending*. Comme les CNNs ne sont pas complètement invariants à la rotation, les performances devraient être meilleures. Si ce n'est pas le cas, c'est qu'un blending parfait dans le contexte est vraiment nécessaire.

Apport du RPN. Comme dans l'étude de Faster R-CNN, l'ajout de nombreuses différentes voitures ne peut avoir qu'un effet positif sur le premier étage de propositions de régions. Il serait intéressant d'entraîner un RPN fixe sur une base hybride pour ensuite fine-tuner la partie Fast R-CNN sur la BDD Munich originale, nous nous rapprocherions dans ce cas là des travaux de [99]. Mais notre but premier est d'avoir un générateur de base telle qu'il ne soit pas nécessaire de l'adaptation de domaine en aval. C'est peut être trop ambitieux, mais les récents travaux sur les GANs sont encourageants.

BDD Munich, les voitures dans leur contexte. Même si la cohérence globale importe peu, une voiture trop brillante dans un endroit à l'ombre serait complètement irréaliste. De même qu'une voiture trop sombre dans un endroit en plein soleil. Les techniques de blending usuelles permettent, dans une certaine mesure, de pallier aux irrégularités de contraste et de couleur. Mais ne serait-il pas bénéfique d'utiliser encore des GANs, connus pour être bien meilleurs par construction pour tromper d'autre CNNs. C'est l'idée de GP-GAN [246] qui utilise une pyramide laplacienne comme le blending laplacien pyramidal de la section précédente (idée qui est aussi présente dans LAP-GAN [46]), ainsi que les équations du blending de Poisson pour apprendre à mélanger proprement des images dans des fonds texturés.

Discrimination fine entre classes, le problème de la BDD Munich. Un des grands problèmes de ShapeNet vis-à-vis de Munich et que la définition des voitures de Munich est très restreinte et peu variable, celle de Shapenet au contraire est très large (voir loufoque parfois). Il faudrait alors préalablement à l'utilisation des voitures ShapeNet définir un protocole pour se débarrasser des voitures trop différentes des mo-

dalités de la BDD Munich. Nous pouvons faire un premier choix en utilisant les tags des différentes voitures pour se débarrasser des speeders et autres voitures peu communes. Une inspection visuelle peut aussi être nécessaire pour se débarrasser des voitures aberrantes. Nous pouvons aussi nous baser sur la forme des masques pour éliminer les voitures trop allongées ou trop grosses. Bien sûr la décimation des objets de ShapeNet dépend des applications.

5.5 Conclusions

Nous avons pu dans ce chapitre étudier la génération de vignettes d'images réalistes et toutes les problématiques liées à leur intégration dans un contexte. Nous avons montré quantitativement les avantages de l'approche GAN pour la création d'images réalistes. À partir de ce nouveau mécanisme, il est maintenant possible de créer des bases quasi infinies d'un réalisme soutenu. La méthode utilisée possède malgré tout des limitations dues aux artefacts de blending et à la pertinence des objets nouvellement créés vis-à-vis de l'entraînement d'un détecteur, et ne nous a pas permis d'améliorer les performances des détecteurs.

Chapitre 6

Conclusions et perspectives

Cette thèse nous aura permis de comprendre en profondeur, par des ré-implémentations et des améliorations, la plupart des méthodes de l'état de l'art en détection d'objets. Elle aura permis également la réalisation de méthodes originales. Le déploiement de ces méthodes sur notre problème spécifique de la détection de petits véhicules en imagerie aérienne fut l'occasion de mettre en exergue leurs limitations et ainsi de trouver des axes de recherche. L'orientation de la dernière année de thèse, complémentaire aux autres méthodes développées, nous aura donné une compréhension plus fine encore de la notion d'objets visuels.

La liste des publications effectuées au cours de cette thèse est située en Annexe E.

6.1 Sélection d'exemples pour l'apprentissage de CNN

6.1.1 Contributions

Nous avons montré qu'il était possible, en surveillant l'entraînement d'un CNN grâce à des stratégies de bootstrapping et la gestion du déséquilibre de classe, d'arriver à des performances état de l'art sur une base complexe. Cela fait écho aux travaux récents [212, 213] montrant des performances état de l'art sur des BDD plus classiques *sans pré-entraînement ImageNet*.

6.1.2 Limitations et pistes d'amélioration

Ces stratégies sont cependant lourdes (re-création de la base de vignettes après chaque passe de bootsrapping) et pourraient peut-être être réalisées "en ligne" en utilisant l'entraînement complètement convolutionnel du chapitre 2 (la carte obtenue permet de sélectionner comme avec OHEM [215] les exemples les plus mal classés). De plus le CNN utilisé est peu profond donc a moins d'expressivité et de pouvoir de discrimination. Un réseau à un étage à la YOLO ou SSD spécifiquement conçu pour l'imagerie aérienne pourrait être construit en s'inspirant de l'existant comme PVANet [128] ou SqueezeDet [245] créés pour des applications liées aux voitures autonomes. De plus, des techniques de quantification, de pruning et de choix de filtres ou de structures peuvent encore accélérer les réseaux et réduire encore l'empreinte énergétique et la complexité du détecteur résultant [80, 186, 111, 265, 113, 112, 107, 110, 259, 174], parmi lesquels on trouve dans le désordre des réseaux avec poids binaires ou quantifiés sur peu de bits, des réseaux avec moins d'opérations, des réseaux compressés, etc.

6.2 Estimation d'orientations avec le framework de régression de distribution à distribution

6.2.1 Contributions

Nous avons montré les avantages de voir le problème de la régression comme un problème de classification et plus encore comme un problème de régression de distribution à distribution (D2D). Cela permet de s'affranchir de certains effets négatifs du coût utilisé en régression classique. Nous avons aussi montré qu'une bonne estimation d'angle permettait de construire une cascade, en utilisant cette information pour rectifier les détections, et ainsi gagner presque 10 points d'AP.

6.2.2 Limitations et pistes d'amélioration

Avec plus de temps, nous aurions pu intégrer le coût D2D dans notre framework Faster RER CNN et observer ainsi probablement des gains en performances au moins

sur l'estimation de la rotation.

6.3 Faster RER CNN et Align-Faster RER-CNN

6.3.1 Contributions

Nous avons adapté un pipeline de détection complexe pour incorporer nos trouvailles. Cette transformation a nécessité beaucoup d'efforts de réflexion avec notamment des implémentations C++ multithreadées et CUDA. Nous avons proposé : une nouvelle paramétrisation de boîtes tournantes, un RoI-pooling tournant, une fonction de coût de régression robuste ainsi que des moyens simples de généraliser le calcul d'IoU de manière vectorisée sur des boîtes tournantes.

6.3.2 Limitations et pistes d'amélioration

Les récents travaux sur le pooling dépendant de la position de R-FCN [41] ainsi que sur le pooling déformable [166] permettent facilement d'étendre notre framework. Avec l'interpolation bilinéaire, il est facile de donner un degré de liberté supplémentaire à la position de chaque cellule *en conservant le même angle pour chacune des cellules d'une région* pour respecter la structure d'une voiture. Ce faisant peut être gagnée encore plus de discrimination en découvrant, de manière non supervisée, des parties d'objets comme les roues. Pour compléter plus exhaustivement il serait intéressant de conduire des expériences intégrant les avancées récentes comme les FPN [144] ou bien HyperNetwork [132] pour gagner en résolution. Nous pourrions aussi, de manière encore plus simple, extraire les propositions du RPN tournant à la manière d'un R-CNN pour ré-échantillonner les véhicules détectés sans perdre trop d'informations comme dans [261]. Les ancres méta de [255] pourraient permettre de s'aligner sur les applications visées en choisissant des a priori utilisateurs (ancres) dépendant de la situation à l'inférence.

6.4 Génération de bases synthétiques infinies pour l'apprentissage de détecteurs

6.4.1 Contributions

Nous avons construit un nouveau framework TriCycleGAN permettant de générer des grandes bases d'un réalisme intermédiaire entre les bases réelles et les véhicules "cartoon", et ce de manière automatique.

6.4.2 Limitations et pistes d'amélioration

Les détecteurs entraînés sur ces bases augmentées ne sont pas meilleurs que les détecteurs appris entièrement sur la base réelle. Il faudrait se focaliser sur le blending (par exemple s'inspirer de [246]) ainsi que sur l'alignement des descripteurs des images synthétiques au niveau local. Les travaux sont en cours. De plus, les approches de transfert de style [64] peuvent aussi donner des résultats visuellement intéressants (voir SIMS [181] par exemple) sans forcément avoir la complexité de notre pipeline actuelle.

6.5 Établissement d'un état de l'art complet des détecteurs à base de CNNs

6.5.1 Contributions

Nous avons rédigé un "survey" (enquète) de 60 pages, en cours de soumission, contenant plus de 450 citations d'articles donnant un état de l'art complet ainsi que des pistes de recherches abouties et non abouties donnant forme au futur de la détection d'objets ; nous avons aussi listé dans ce travail tous les challenges qui semblent encore résister au CNNs.

6.5.2 Limitations et pistes d'amélioration

Le workshop COCO de ECCV2018 se déroulant pendant l'écriture de ces lignes, il semblerait que, bien que les architectures de type Faster R-CNN soient encore très

présentes, le gagnant de la compétition de détection utilise une nouvelle technique mystérieuse n'utilisant pas de grille d'ancres dense. Peut-être est-ce le signe d'une possible révolution en détection d'objets ?

Annexe A

Anatomie des CNNs

Cette annexe présente les réseaux de neurones et plus particulièrement les CNNs.

A.1 Réseau de neurones classique (NN)

Un neurone réalise une fonction mathématique simple sur ses entrée(s) pour produire une sortie scalaire. En général, la quasi totalité des neurones standards applique une fonction non linéaire f appelée activation (les premières fonctions utilisées étant des seuils durs, le neurone n'était donc non-nul ou activé que si la somme de ses entrées dépassait une valeur seuil) sur une somme pondérée de leurs entrées :

$$o = f(\sum_{i=1}^{n} w_i * e_i) \tag{A.1}$$

On peut aussi rajouter un biais b ce qui est équivalent à ajouter un 1 sur le vecteur d'observation x. L'importance de chacune des entrées est donc modulée par un poids w, paramètre apprenable du neurone. Il est donc tout à fait normal d'imaginer connecter des neurones entre eux formant ainsi un réseau. Une entrée de dimension d peut donc être connectée à n_0 neurones créant $d * n_0$ connexions ou poids. Ces neurones peuvent être à leur tour connectés à n_1 neurones créant $n_0 * n_1$ connexions et ainsi de suite jusqu'à une sortie de dimensionnalité voulue. Le cardinal de l'indice de n indique le nombre de couches du réseau de neurones. Pour écrire la fonction du réseau de neurones, on utilise

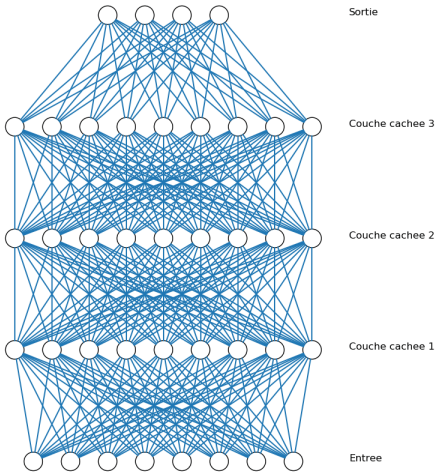


FIGURE A.1 – Structure d'un réseau de neurones

le formalisme matriciel (voir tensoriel) :

$$NN_{W_1,...,W_n}(x) = f(...(f(f(x * W_1) * W_2) * ...) * W_n) \tag{A.2}$$

Où x est une matrice (N,D) représentant les N exemples de dimension D de l'ensemble d'apprentissage, W_1 la matrice de poids de la première couche de dimension D par N1. W_k de dimension N par N_k, et f appliquée coefficient par coefficient. On retrouve bien en regardant la matrice $C1 = f(x * W_1)$, $C1(n, n1) = f(\sum_{d=1}^{D}(x_n)_d w_{dn1})$ chaque équation de neurone individuel appliqué à un exemple de l'ensemble d'apprentissage avec w_{dn1} le poids connectant la composante d de x au neurone n1. La figure A.1 en présente un exemple. Ainsi, si on veut faire converger la sortie du réseau de neurones vers une cible fixe pour chaque exemple par descente de gradient, on doit calculer le gradient d'une fonction composée de multiplications et de fonctions non linéaires. Pour calculer le gradient de manière efficace, on calcule toutes les activations sur l'ensemble d'entrainement de chaque neurone puis on back-propage l'erreur pour savoir localement comment bouger chaque poids individuel pour faire descendre le coût de 1. Cela correspond à des multiplications de matrices jacobiennes que l'on ne développera pas ici mais dont on voit

un exemple dans le Chapitre 2. Les coefficients des matrices jacobiennes sont calculés à partir des activations de manière analytique pour chaque brique du réseau de neurones (backward).

L'optimisation est dans la plupart des cas réalisée par descente de gradient stochastique (SGD) par mini-batch ce qui revient à approximer le coût sur l'ensemble des exemples d'apprentissage par une somme partielle sur des exemples choisis. Lorsqu'il n'y a qu'un seul exemple par mini-BATCH, on parle de SGD. Lorsqu'il y en a un nombre n $\ll$ N, on parle de mini-batch SGD et si $n = N$, on parle de batch SGD. Comme N est très grand dans la plupart des cas, on utilise SGD ou mini-batch SGD pour des questions de calcul. De plus SGD avec un seul exemple par itération étant trop bruitée on préfère de grands mini-batchs de taille multiple de 32 pour faciliter le passage sur GPU (256 pour les réseaux classiques en général).

A.2 Réseau de neurones convolutionel (CNN)

Si l'on voulait mettre en entrée de notre réseau de neurones classique (à une couche cachée par exemples) une image, on perdrait alors toute structure spatiale ainsi que toute structure de couleur, ce qui obligerait le réseau à une gymnastique compliquée lors de l'apprentissage du fait de la multiplicité des poids ainsi que de l'aspect non ordonné de l'entrée. On obtiendrait alors une fonction pouvant être arbitrairement proche pour des images données de n'importe quelle sortie (avec suffisamment de neurones [104]) mais probablement inutilisable car non générale.

L'idée de Yann LeCun est de forcer des a-priori structurels sur les images dans le corps du réseau de neurones.

Il impose de réaliser des sommes *locales* de poids sur des voisinages carrés de dimension faible vis à vis de la taille de l'image en réutilisant *les mêmes poids* à différentes positions sur l'image (weight sharing). Ainsi, non seulement on gagne la prise en compte de la spatialité de l'entrée mais en plus on diminue le nombre de poids donc on régularise le réseau. La convolution réalise exactement cette opération. De plus, il organise des convolutions 2Ds en profondeur pour appliquer différents filtres sur différentes couleur de

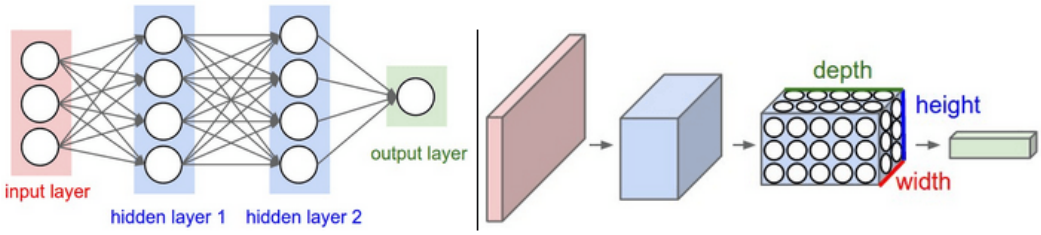


FIGURE A.2 – NN vers CNN tiré de [124]

l'image. Les différentes couches obtenues restent de la même forme spatiale que l'image. On les appelle des descripteurs "carte" ou cartes convolutionelles ("feature maps"). La transformation d'un réseau de neurones classique en un CNN est représenté Figure A.2. Ces cartes sont, en général, agrégées spatialement et localement par des modules de pooling ce qui diminue leur dimension spatiale. Lorsque la dimension spatiale est devenue contrôlable (7x7 en général) on continue sur un réseau de neurones classique jusqu'à la sortie. Le formalisme utilisé habituellement fait appel à des matrices de dimensions supérieures à 2, c'est-à-dire à des tenseurs, jusqu'à arriver aux couches complètement connectées où l'on retrouve des matrices.

A.3 Les composantes classiques d'un CNN

A.3.1 Les filtres

Les filtres utilisés dans la littérature sont généralement carrés. La taille la plus utilisée est 3x3 popularisée par VGG-16 [217] mais on trouve aussi des filtres 5x5 et 7x7 dans AlexNet [135] ou bien encore dans les réseaux Inceptions [225]. La dimension des filtres est impaire pour des raisons d'alignement entre les entrées et les descripteurs cartes : le résultat de l'application d'un filtre se retrouve sur le pixel central de celui ci, si le filtre n'a pas de pixel central cela introduit forcément un décalage, quelle que soit la convention adoptée. En général on ajoute des bords nuls à l'image symétriquement pour garder la même taille spatiale en entrée et en sortie d'une convolution. La relation de taille pour une convolution d'un filtre sur un descripteur carte est la suivante :

$$o_{out} = \frac{o_{in} - k + 1}{s} \tag{A.3}$$

Ainsi, pour un filtre 3x3, on rajoute une ligne de zéros de tous les côtés du descripteur, le centre du filtre aux bords se retrouve à l'emplacement du pixel considéré. Pour initialiser les filtres, avant de lancer l'optimisation, différentes techniques ont été proposées, la plupart proposant d'initialiser les filtres à des valeurs proches de 0 ([73, 89]) et les biais à 0. On peut trouver des schémas d'initialisation plus sophistiqués comme Saxe [205], qui utilise la décomposition en valeur singulières des filtres applatis ou [249] utilisant aussi des filtres orthogonaux .

A.3.2 Les activations

Les activations sont des fonctions forcément non linéaires sinon on pourrait réduire le réseau à une seule couche en chaînant les matrices. Les premières à être utilisées furent la sigmoïde et la tangente hyperbolique, mais elles entrainaient des problèmes d'explosion ou de disparition du gradient et sont abandonnées maintenant en tout cas en détection. Encore que l'on trouve récemment des articles théoriques les utilisant [249]. L'activation ReLU [169] et ses variantes (Leaky ReLU [156], PReLU [89], RRelu [252]) dominent complètement. Cependant d'autres alternatives comme ELU [33] ou SELU [131] ont fait leur preuves. A priori toute fonction sous-différentiable peut etre utilisée. Par exemple des sinus ou cosinus ont été utilisés pour des applications précises [62] mais il est recommandé d'utiliser des ReLU pour commencer. La recherche de fonctions d'activation étant un domaine très prolifique il ya quelques années ralentit un peu car il est de plus en plus facile d'entrainer des CNNs, même si on peut citer la récente Swish [185].

A.3.3 Le pooling

Après avoir obtenu un descripteur carte on réalise des statistiques locales non paramétrables. En prenant en général le maximum sur des cellules 2x2 disjointes [140] même si AlexNet [135] introduit des cellules se recouvrant pour lutter contre le surapprentissage. D'autres types de pooling ont été proposés comme l'average pooling utilisé dans ResNet [90]. Le fractionnal pooling [76] introduit des cellules de taille variable stochastique.

A.3.4 Les coûts

Lorsqu'on entraine un CNN en classification, on utilise majoritairement l'entropie croisée ou la distance de Kullback-Leibler (à une constante près) [13] entre la sortie du réseau et la sortie voulue qui peut être assimilée à une distribution. Le softmax impose d'avoir la somme des sorties égale à 1.

$$C(p) = -p * log(softmax(l)) \tag{A.4}$$

Où $*$ désigne le produit scalaire, l la sortie du réseau et p la cible de même taille. Dans le cas d'une classification binaire comme dans le cas voitures vs fond, ou bien objet vs fond, on peut aussi utiliser la sigmoïde :

$$C(z) = z * -log(\sigma(x)) + (1 - z) * -log(1 - \sigma(x)) \tag{A.5}$$

Dans ce cas, z est un scalaire. C'est aussi ce qu'on utilise quand les classes ne sont pas mutuellement indépendantes, on entraine n classifieurs et on fait l'argmax indépendamment sur les n classifieurs : une image peut être à la fois une voiture et un pick-up. Le coût de charnière (Hinge-loss) utilisé dans le SVM [36] est beaucoup moins répandu car il donne de moins bons résultats en général avec les CNNs en classification sauf dans le cas des réseaux siamois [16] et triplets [209] où il est généralement préféré. Pour la régression on utilise normalement un coût moindre carré dont on adoucit les gradients lorsqu'ils sont trop élevés en utilisant le coût de Huber [13]. Mais tous les coûts sous-différentiables sont possibles et leur choix dépend bien souvent de l'application et de la performance empirique du modèle dans le cas envisagé.

A.3.5 Dropout et dropconnect

Pour régulariser le réseau, principalement dans les réseaux de neurones classiques et les couches complètement connectées qui comportent beaucoup de paramètres, deux stratégies ont été proposées. La plus utilisée, le dropout [100] éteint avec une probabilité p des neurones du réseau pendant l'apprentissage créant ainsi un sous réseau plus fin.

Ainsi le réseau apprend à ne pas trop se reposer sur un neurone en particulier. Pour le test, comme on a entrainé le réseau avec $(1-p)*N$ neurones en moyenne, on normalise la sortie des couches pour tenir compte de cet effet en multipliant chaque neurone par (1-p). La seconde DropConnect [240] est moins brutale et se débarasse de connexions entre neurones et non plus de neurones entiers.

A.3.6 Les modules Inceptions

Googlenet [226] introduit un nouveau module plus complexe que les architectures classiques à la VGG [217]. Il combine différents champs réceptifs tout en gardant un budget raisonnable en termes de poids et de calculs grâce aux convolutions 1 par 1. Les sorties de ces différents filtres sont ensuite concaténées. Un exemple d'un tel module est explicité ci-dessous en utilisant une syntaxe inspirée de keras, qui est proche du pseudo-code :

```
branch_0 = Conv2D(64, (1,1), padding='same', activation='relu')(input_img)


branch_1 = Conv2D(64, (1,1), padding='same', activation='relu')(input_img)
branch_1 = Conv2D(64, (3,3), padding='same', activation='relu')(branch_1)

branch_2 = Conv2D(64, (1,1), padding='same', activation='relu')(input_img)
branch_2 = Conv2D(64, (5,5), padding='same', activation='relu')(branch_2)

branch_3 = MaxPooling2D((3,3), strides=(1,1), padding='same')(input_img)
branch_3 = Conv2D(64, (1,1), padding='same', activation='relu')(branch_3)

output =concatenate([branch_0, branch_1, branch_2, branch_3], axis = 3)
```

GoogleNet est aussi le premier à utiliser des coûts auxiliaires aux couches plus basses du réseau pour mieux pallier à la disparition du gradient, précurseur des connexions skips de la section suivante. La seconde itération des modules Inception [227] remplace les

convolutions 5x5 par deux convolutions 3x3 de champs réceptifs équivalents $(3+(3-1) = 5$ voir chapitre 2) mais avec là encore moins de paramètres et applique cette astuce pour combiner encore plus de champs réceptifs avec un budget limité.

A.3.7 La normalisation de batch

Un article très influent [115] propose une normalisation des activations de chaque couche pour que le réseau ne soit pas constamment en train de s'adapter à différentes distributions d'activations à chaque différent mini-batch. A chaque couche d'activation, on soustrait sa moyenne sur le mini-batch et on la divise par la variance sur le batch afin d'avoir sur le mini-batch une couche centrée réduite. Deux paramètres seulement par couches permettent ensuite d'apporter un facteur d'échelle et un biais à cette distribution si besoin. Cette astuce accèlère énormément l'apprentissage. Cependant, il n'est pas encore vraiment clair pourquoi.

A.3.8 Les connexions saute-moutons et denses

[90] remarque qu'entraîner des réseaux de profondeur croissante n'augmente pas les performances sur l'ensemble d'apprentissage mais au contraire les réduit ce qui est contre-intuitif car les réseaux de profondeur moindre ont moins de capacité que ceux plus profonds. Il y a donc un problème d'optimisation des réseaux trop profonds. Lorsqu'il y a trop de couches les premières couches ne reçoivent que très peu de gradients, des couches proches de la sortie (entre autres). Pour remédier à cela, l'auteur propose d'introduire des connexions identitaires entre des couches basses et des couches hautes *de même dimension* en les combinant par une addition (apprentissage résiduel) ou une concaténation :

$$y = activation(x + CNN(x)) \tag{A.6}$$

Où CNN est un enchainement de CNNs classique. L'article DenseNet [108] propose d'élargir le concept en connectant sur une longueur donnée toutes les couches entre elles : chaque couche prend toutes les précédentes du bloc en entrée. Toutes les entrées sont alors concaténées, cela impose d'avoir la même dimension spatiale pour toutes les

cartes convolutionnelles dans le même bloc c'est pourquoi plusieurs blocs denses sont connectés par des convolutions et du pooling traditionnel pour composer un DenseNet. On peut aussi citer le mélange des modules Inceptions et des connexions skips [225].

A.3.9 Généralisation des CNNs

Le plus étonnant dans les CNNs est qu'ils ne semblent pas respecter les principes de BIC et d'Akaïké [13], qui pénalisent les modèles comportant trop de paramètres. Leur nombre immense de paramètres devrait normalement les rendre très mauvais en généralisation, mais il semble que grâce à l'a-priori de structure très fort des convolutions, cela suffise à limiter la capacité des modèles et à les rendre généralisables, même si cette explication est insuffisante comme l'a montré [258]. Quelques articles dans le désordre sur le sujet évoquent tantôt le fait que les minimas soient plats ou non, tantôt la théorie de l'information, tantôt l'effet régularisateur des mini-batch : [258, 216, 48, 127, 142, 206]. Les résultats se contredisent et la seule véritable conclusion que l'on peut avoir à ce stade et qu'il y a quelque chose de très fondamental dans les CNNs et/ou leur entrainement qui doit expliquer leurs capacités exceptionnelles sur les images (ainsi que dans d'autres domaines structurés comme la compréhension du langage parlé).

A.3.10 Divers

Nous ajoutons ici quelques références choisies non indispensables pour la thèse mais intéressantes pour ceux souhaitant approfondir leurs connaissances sur les CNNs en classification d'images :

— les modules de transformation spatiale[118] qui permettent de ré-échantillonner l'image pour avoir une pose normalisée
— le dropout à l'inférence [60] pour avoir une modélisation des incertitudes du réseau
— les réseaux de profondeur stochastique [109] ou de manière similaire au dropout on éteint pendant l'entrainement des couches entières de neurones ce qui est rendu possible par les connexions skips.
— Les modules d'agrégations de ResNext [250]
— Les modules Squeeze and Excitation [106] qui ont remporté la dernière itération

d'ImageNet en classification en introduisant une branche d'average pooling par chaîne (Squeeze) puis un réseau complètement connecté mélangeant l'information des différentes chaines pour finalement multiplier chaîne par chaîne la couche précédente (Excitation).

- La régularization Shake-Shake [63] qui propose des réseaux à 3 branches dont les sorties sont sommées de manière stochastique, ShakeDrop[254] qui étend cette stratégie au réseau comportant une seule branche.
- Le CutOut [47] qui consiste à masquer des parties des images d'entrainement de manière aléatoire
- AutoAugment [39] qui apprend des stratégies de data augmentation par apprentissage par renforcement
- La recherche automatique d'architectures par des approches d'apprentissage par renforcement et évolutives [270, 179, 148, 146, 191]

On voit ici l'inventivité des chercheurs ainsi que le royaume de possibilités qu'offrent les CNNs.

Annexe B

Formalisme GAN et VAE

Cette annexe présente deux modèles génératifs populaires à base de CNNs dont la connaissance est utile pour la thèse : les VAE [130] et les GANs [75].

B.1 Auto-encodeurs variationnels (VAE)

Toute cette section donne les explications nécessaires à la bonne compréhension des VAEs de Kingma et Welling [130]. On cherche à estimer les paramètres θ^* d'une distribution paramétrique $p^*_\theta(x)$ expliquant notre ensemble d'apprentissage X. En outre on veut être capable de générer de nouveaux exemples de cette distribution. Enfin la log-vraisemblance est d'une forme trop complexe pour la résolution analytique. Comme pour EM [13], on suppose alors l'existence de variables cachées/latentes/non observées/non connues/code z dont la connaissance permettrait de rendre l'optimisation de la log-vraisemblance plus facile. (juste pour fixer les idées, il peut s'agir de la mise en correspondance de toutes les données de l'apprentissage à leur gaussiennes respectives par exemple dans le cas d'un mélange de gaussiennes [13]). On fait en plus l'hypothèse que nos données x sont générées selon le processus suivant : on génère z aléatoirement puis une fonction déterministe donne x. La connaissance de la densité de z notée $p_{\theta^*}(z_i)$ et de la densité conditionnelle $p_{\theta^*}(z|x)$ permettrait d'obtenir non seulement $p(x)$ par intégration mais aussi probablement de générer de nouveaux x ; Le problème est que nous ne connaissons aucune de ces quantités et z n'est pas accessible. En plus, la den-

sité postérieure $p_{\theta^*}(x|z)$ n'est pas calculable ce qui nous aurait permis en utilisant EM d'optimiser sur θ à z fixé. On commence par écrire la log-vraisemblance :

$$logp(x_1, ..., x_n) = \sum_{i=1}^{n} logp(x_i) \tag{B.1}$$

On va essayer de réécrire $logp(x_i)$ sous une forme plus simple, on considère d'abord l'expression suivante (pour n'importe quelle distribution q sur Z) :

$$E = D_{KL}(q||p_\theta(z|x^i)) + \mathcal{L}(\theta, x_i, q) \tag{B.2}$$

avec $\mathcal{L}(\theta, x_i, q) = \mathbb{E}_{z\sim q(z)}(log(\frac{p_\theta(x^i,z)}{q(z)}))$. On développe en utilisant la définition de la distance de Kullback :

$$E = \mathbb{E}_{z\sim q(z)}(log(\frac{q(z)}{p_\theta(z|x^i)}) + log(\frac{p_\theta(x^i, z)}{q(z)})) \tag{B.3}$$

On utilise la règle de Bayes pour simplifier $p_\theta(z|x^i)$:

$$E = \mathbb{E}_{z\sim q(z)}(log(q(z)) - log(\frac{p_\theta(x^i, z)}{p_\theta(x^i)}) + log(p_\theta(x^i, z)) - log(q(z))) \tag{B.4}$$

On simplifie :

$$E = \mathbb{E}_{z\sim q(z)}(log(p_\theta(x^i)) \tag{B.5}$$

Or l'expression dans l'espérance ne dépend pas de z. On peut donc la sortir de l'espérance. Finalement :

$$E = log(p_\theta(x^i)) = D_{KL}(q||p_\theta(z|x^i)) + L(\theta, x_i, q) \tag{B.6}$$

On a donc décomposé la quantité à calculer sous la forme d'un terme mystérieux $\mathcal{L}$ et d'une distance qui est toujours positive (par Jensen). $\mathcal{L}$ *est donc une borne inférieure pour la log vraisemblance d'une donnée.* On l'appelle la borne inférieure variationnelle. Si on veut maximiser la log-vraisemblance, il suffit donc de maximiser $\mathcal{L}$ (même si rien ne dit si $\mathcal{L}$ est proche ou non de la log-vraisemblance). Pour minimiser $\mathcal{L}(\theta, x_i, q) = \mathbb{E}_{z\sim q(z)}(log(\frac{p_\theta(x^i,z)}{q(z)}))$, on utilise exactement les mêmes astuces (la règle de Bayes) pour

en trouver une expression plus simple :

$$\mathcal{L}(\theta, x_i, q) = -D_{KL}(q(z)||p_\theta(z)) + \mathbb{E}_{z\sim q(z)}(logp_\theta(x^i|z)) \tag{B.7}$$

Pour maximiser cette expression, il faut donc rapprocher une distribution q de l'a-priori sur la distribution de z, que nous choisissons en général gaussien centré réduit ($p_\theta(z) \sim \mathcal{N}(0, I)$). Pour q, on choisit une distribution conditionnelle par rapport à x_i : $q(z|x^i)$. C'est ce que nous appellerons le modèle de reconnaissance de $p_\theta(z|x^i)$ (du fait de l'équation B.1), qui va servir d'approximation pour cette vraie distribution, que l'on ne peut pas évaluer directement. Pour calculer le gradient de L par rapport à q on utilise l'astuce de reparamétrisation suivante : Pour toute probabilité conditionnelle respectant certaines conditions (voir [130] au besoin) on a :

$$z \sim q(z|x^i) \Leftrightarrow z = e_\phi(\epsilon, x^i) \; avec \; \epsilon \sim p(\epsilon) \tag{B.8}$$

Avec e_ϕ une fonction déterministe différentiable (on se doute qu'on utilisera un réseau de neurones). Un exemple d'une telle fonction est celle qui calculerait la moyenne et la variance d'une gaussienne dépendant de x^i. Ainsi, on peut maximiser le premier terme de L. Pour le second, puisqu'on peut aussi générer les z à partir de x_i avec l'astuce de reparamétrisation, il suffit d'utiliser un second réseau de neurones $d_\delta(z)$ pour approximer la fonction donnant x_i à partir d'un z tiré selon la conditionnelle $e_\phi(\epsilon, x^i)$. Ainsi on aura successivement encodé et décodé x_i. La fonction à optimiser par ascension de gradients est donc (en sommant sur les $\mathcal{L}$ associés à chaque exemple d'apprentissage sur un mini-batch comme classiquement) :

$$C_{VAE}(\phi, \delta) = -\sum_{i=1}^{N} D_{KL}(q(z|x^i)||p_\theta(z)) + \frac{1}{L}\sum_{i=1}^{N}\sum_{l=1}^{L} logp_\theta(x^i|z^{i,l}) \tag{B.9}$$

où la variable L représente le nombre de tirages de ϵ et est un hyper-paramètre (la valeur de 1 pour L semble convenir selon [130]). D'un point de vue très haut niveau, on peut donc voir ça comme maximiser le coût de reconstruction d'un auto-encodeur stochastique tout en forçant le code à respecter un a-priori d'où le nom d'auto-encodeurs

FIGURE B.1 – Exemples de génération d'exemples pour des VAE avec des codes de dimensions de gauche à droite de : 2, 5, 10 et 20. (tirés de l'article [130])

variationnel.

Si l'a-priori choisi sur le code est gaussien et qu'on choisit d'encoder la variance et la moyenne de q que l'on prend aussi gaussienne, le premier terme a une expression analytique très simple *sans approximation* (z disparait dans le calcul de l'espérance). Quant au second terme il est possible de le voir *comme une entropie croisée entre les vrais pixels des images et les pixels générés obtenus par décodage de z* où les pixels sont analogues aux labels de l'apprentissage supervisé classique (cas Bernoulli [130]). Quelques exemples de VAE sont présentés Figure B.1.

B.2 Réseaux de neurones génératifs adversariaux (GANs)

Goodfellow et al. [75] propose d'entrainer simultanément deux réseaux avec des objectifs contraires (d'où le nom d'adversaires). Le premier, le modèle génératif à proprement parler (nous l'appellerons G), doit produire des images à partir d'un bruit aléatoire respectant la distribution des vraies images, le second est un juge D, qui doit trouver les images "fausses" (générées par G) parmi celles qu'il reçoit. Cette formulation est intéressante pour plusieurs raisons. La première est qu'elle n'optimise pas directement une log-vraisemblance comme la distance de Kullback-Leibler des images produites aux vraies images car cela serait difficile à exploiter (la dimensionnalité de l'espace des images est trop grande pour qu'on ait accès même indirectement à la probabilité réelle des images) mais les gradients sont calculés par l'intermédiaire du juge dont les performances évoluent au cours du temps ce qui est similaire à l'apprentissage par curriculum de [10]. De plus, le fait d'affronter un adversaire introduit une supervision gratuite.

Bien sûr, G et D sont paramétrisés par des réseaux de neurones (MLP), et appris avec les mêmes techniques (back-propagation et mini-batch SGD). La formulation min-max originale du problème fait intervenir l'équation suivante :

$$\min_G \max_D(C(D,G)) = \min_G \max_D(\mathbb{E}_{x\sim p_{vraie}(x)}(log(D(x))) + \mathbb{E}_{z\sim p_z(z)}(log(1-D(G(z)))))) \tag{B.10}$$

Il y a cependant quelques problèmes avec cette formulation :

— Comme c'est un min max, il faut alterner les optimisations de G et D, sans que l'un ou l'autre prenne trop l'avantage sur son concurrent. Comme D a une tâche plus facile il converge plus vite s'il est trop bon dès le début. Le problème est trop difficile pour le générateur et les gradients locaux ne permettent pas de trouver un bon optimum, mais à l'inverse s'il n'est pas assez bon alors les gradients qu'il fournit au générateur ne sont pas informatifs et G ne progresse pas. Ces difficultés imposent au papier original une procédure compliquée dans laquelle le juge est optimisé pour un nombre fixé de pas de gradient puis D, pas forcément avec le même taux d'apprentissage, et ce de manière alternative,

— au début de l'apprentissage G est trop mauvais et le travail de D est trop facile donc $log(1-D(G(z))$ sature. On utilise plutôt $-log(D(G(z))$, qui donne les mêmes points fixes (le générateur maximise donc $log(D(G(z)))$.

Ces problèmes expliquent la très grande difficulté d'optimisation de ces machines. Le paramétrage de G et D utilise des simples MLP dans l'article original car les CNNs ne convergeaient pas. Il a fallu attendre un an pour que [182] donne des astuces architecturales et d'optimisation pour obtenir les premiers GANs à base de CNN les DCGANs (Deep Convolutional GANs). Maintenant grâce à d'autres astuces d'apprentissage [202] et aux nombreuses autres fonctions de coût plus robustes comme LSGAN [159], WGAN [8], WGAN-GP [77], CT-GAN [243], HingeGAN [163], Relativistic-GANs [122] ainsi qu'à la Normalisation Spectrale [163, 260], les GANs à base de CNN sont non seulement très faciles à entrainer mais aussi donnent des performances incroyables au niveau visuel. La flexibilité du formalisme permet aussi de conditionner les GANs par rapport à une variable externe (une image ou un code déterministe (une classe)). On parle de

FIGURE B.2 – Exemples de génération d'images conditionnellement à la classe à l'aide de SAGAN entrainé sur Imagenet (tirés de l'article [260])

CGAN (Conditional GANs). On voit un exemple d'un GAN conditionnel impressionnant dans : SAGAN [260] qui est présenté Figure B.2). La capacité des CNNs à générer des structures complexes n'est plus surprenante pour le lecteur maintenant familier avec ces structures. Si les CNNs sont capables de reconnaitre des motifs complexes pourquoi ne pourraient il pas en générer ? Nous pouvons nous en convaincre en faisant une ascension de gradient sur l'espace des pixels en entrée d'un classificateur à poids fixes pour maximiser un score de classe comme l'algorithme DeepDream [167]. Mais le formalisme GAN et les fonctions de coût utilisées permettent un niveau de réalisme bien plus évolué que ce à quoi on pourrait s'attendre.

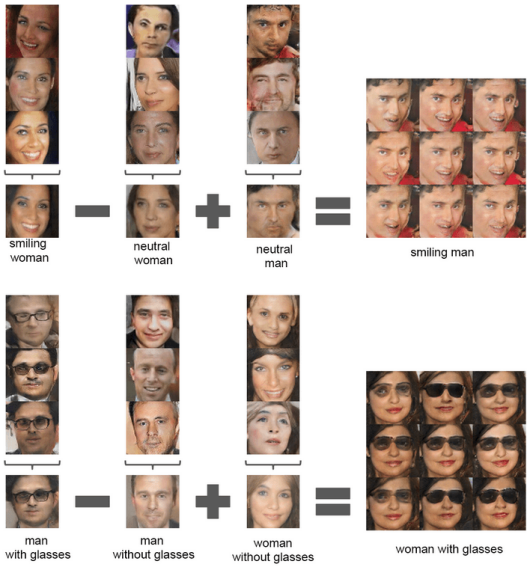


FIGURE B.3 – Exemples d'arithmétique sur les codes latents d'un DCGAN. (tirés de l'article [182])

B.3 Arithmétique sur les variables latentes

Les variables latentes z de dimension fixée intervenant dans les deux modèles sont un peu mystérieuses. Il est possible de montrer empiriquement que quand z est bien choisi et que l'opération se passe bien, chaque composante de z encode une information de nature sémantique sur l'image associée créée par le décodeur/le faussaire. Il est même possible de faire de l'arithmétique sur ces variables pour créer des générations d'images sémantiques cohérentes (à la manière du fameux "$roi - homme = reine$" de word2vec [161])comme présenté sur la Figure B.3. Cela est possible car on reste bien sur la variété des images apprises si l'on oblige z à rester sur le support de l'a-priori sur le code, ce qui permet aussi différentes autres sortes d' interpolations entre images (en utilisant leur z , ce qui est un peu moins facile sur les GANs qui n'implémentent pas l'encodage). Beaucoup de travaux comme Info-GAN [26] cherchent à démêler l'influence de chaque dimension de z pour que cet effet soit magnifié. Il existe bien sûr des travaux qui mêlent les deux formalismes subtilement comme [138, 157, 268].

Annexe C

Résultats visuels de Faster RER-CNN

Il est souvent difficile de juger de la performance d'un détecteur avec seulement des courbes et des chiffres. Parfois les cas d'échec sont intéressants et peuvent révéler des pistes sur comment aller de l'avant. C'est pourquoi nous fournissons :

— des zooms sur des cas intéressants de la BDD Munich

— **Toutes les images** des détections de voiture de **GoogleEarth** qui n'étaient pas déja incluses dans la thèse.

— **Quelques exemples** des détections sur **VeDAI** section C.2

.

Cette annexe se concentre sur Munich du fait de la variété des situations qu'elle présente et parce que la base VeDAI a déjà beaucoup été étudiée dans la thèse.

C.1 Résultats sur Munich

C.1.1 Introduction et code-couleur

On montre des **détections de voiture**, dont les scores sont supérieurs à un seuil. Elles sont en **vert** si leur IoU avec une **voiture** vérité terrain est supérieur à 0.3 (comme Tang et al.) et sinon elles sont **en rouge** si elles n'intersectent aucune autre classe avec IoU¿0.3 ou **en violet** si oui (bus ou camions). On ajoute aussi toutes les vérités terrains

FIGURE C.1 – Un faux positif issu de `2012-04-26-Muenchen-Tunnel_4K0G0160.JPG` et sa version zoomée

FIGURE C.2 – 2 faux positifs de `2012-04-26-Muenchen-Tunnel_4K0G0265.JPG`

non trouvées en **jaune**. Les seuils de détection utilisés sont précisés dans chaque section. Il y en a deux 0.55 et 0.7.

C.1.2 Étude sur les faux-positifs

On présente quelques vignettes zoomées centrées en des faux positifs **avec un seuil de 0.7**.

La Figure C.1 montre un faux positif au milieu d'une forêt. On note qu'il n'y en a qu'un seul alors que la forêt est un milieu très texturé : cela ressemble à une voiture camouflée. La même chose se passe image `4K0G0265` dans la Figure C.2, mais cette fois, il est difficile de dire si les objets trouvés sont des voitures ou non.

Il y a un groupe de faux positifs image `4K0G0130` : visible sur la Figure C.3. La vérité terrain est donc incomplète, car ce sont sans aucun doute des voitures. La plupart des faux positifs sont situés sur des voitures oubliées par l'annotateur ou sur des

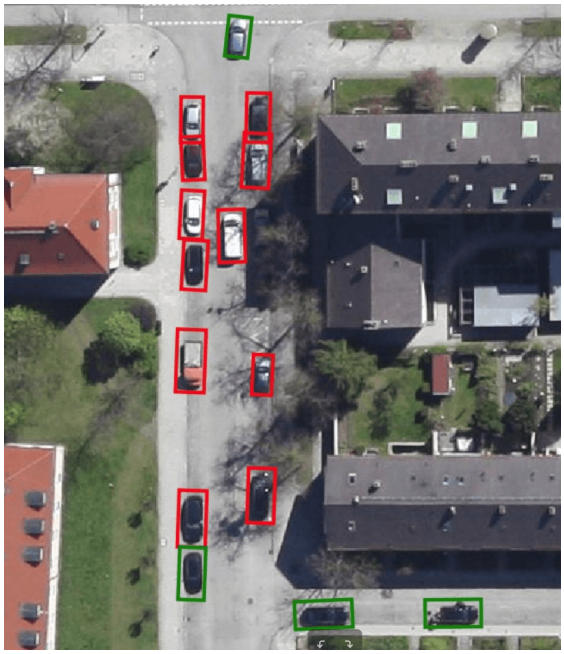


FIGURE C.3 – Grand groupe de voitures détectées comptées comme des faux positifs `2012-04-26-Muenchen-Tunnel_4K0G0130.JPG`

classes similaires. C'est une preuve visuelle, que même si la classification fine est difficile sans stratégie de hard-mining, le RPN remplit son office. La Figure C.4 présente plus d'exemples. Bien sûr même avec un seuil à $t = 0.7$ on peut trouver quelques faux positifs ne reseemblant aucunement à des voitures mais ils sont difficiles à trouver. La Figure C.5 en présente un exemple. Même sur des zones très texturées, comme les forêts ou des zones de chantiers, il y a très peu de faux positifs (voir Figure C.6 et C.1).

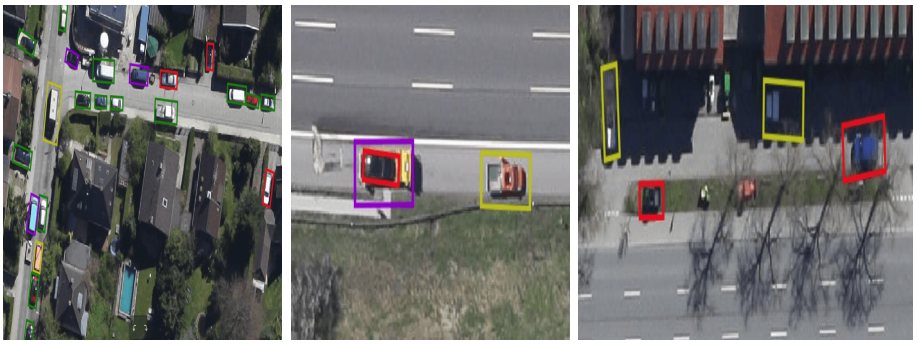


FIGURE C.4 – Divers faux positifs accompagnés de quelques voitures non détectées et de bonnes détections

FIGURE C.5 – Vrai faux-positif

FIGURE C.6 – Zones difficiles sans faux positif

C.1.3 Voitures non détectées

Dans cette section, on présente quelques unes des voitures non détectées **à seuil 0.55**. Comme dans la partie principale de la thèse, nous pouvons regarder les parkings, qui comportent nombre de voitures Figure C.7. On peut voir que le détecteur arrive à gérer les voitures mêmes quand elles sont proches les unes des autres. Malheureusement, les voitures sévèrement occultées sont manquées comme le montre la Figure C.8.

C.2 Résultats sur VeDAI

Pour la base VeDAI il y a de nombreuses classes et folds. On fournit quelques exemples de détections avec un score supérieur à 0.5 sans justifications complémentaires sur la Figure C.10. Cela montre la variété des environnements et contextes présents dans VeDAI (par exemple certains bateaux peuvent être trouvés sur le sol dans des zones urbaines). L'information de couleur est perdue avec l'infra-rouge. Cela rend la tâche encore plus ardue. De plus, la résolution est moindre que pour Munich, ce qui en fait la BDD étudiée la plus proche de nos applications.

FIGURE C.7 – Quelques parkings et les voitures non détectées associées

FIGURE C.8 – Les voitures partiellement occultées ne sont pas détectées

C.3 Résultats sur GoogleEarth

On présente les 3 dernières images (sur 5) non encore incluses dans la thèse. On ne montre que les détections avec un score supérieur à 0.5 Figure C.9. Beaucoup de voitures sont manquées dans les zones ombragées mais il est difficile de généraliser avec si peu d'exemples.

FIGURE C.9 – Le reste des détections de Faster RER-CNN sur GoogleEarth

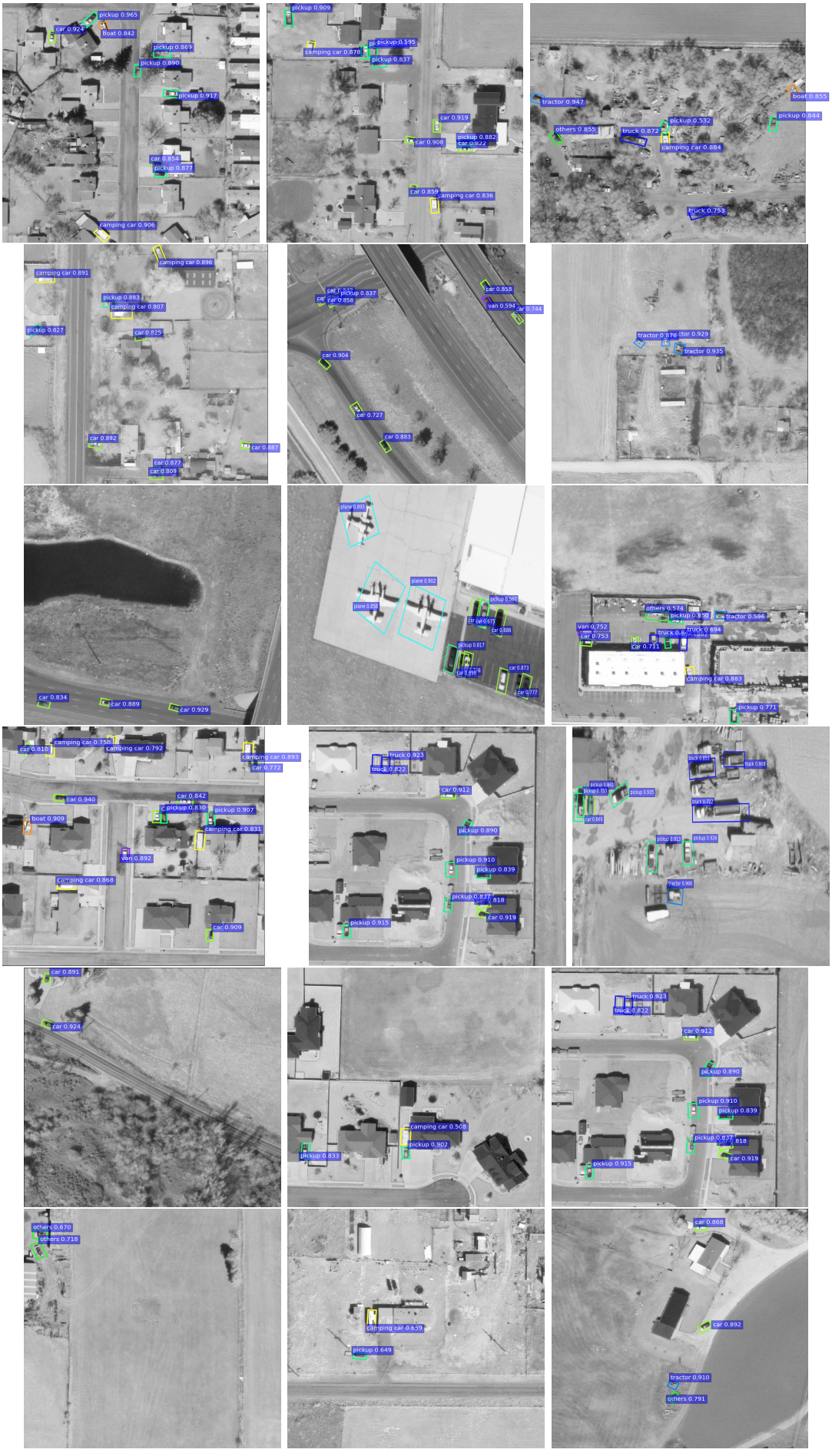


FIGURE C.10 – Quelques exemples de détections de Faster RER CNN sur de VeDAI

Annexe D

Optimisations du code de Sutherland-Hogman

```
#On initialise la sortie au polygône à clipper
outputList = subjectPolygon
#On récupère le premier coin du polygone clippant
cp1= lastElement(clipPolygon)
cp10=cp1[0]
cp11=cp1[1]
for cp2 in clipPolygon:
   #On prend le deuxième coin du polygône clippant ce qui nous donne notre coté clippant
   cp20=cp2[0]
   cp21=cp2[1]
   #inputList est notre polygône à clipper issu de l'étape précédente
   inputList = outputList
   #outputList va contenir le polygône clippé par le côté courant pour l'instant il n'y a rien
   outputList = emptyList
   #On prend le dernier coin du polygône à clipper s
   s = lastElement(inputList)
   s0=s[0]
   s1=s[1]
   #Critère sur le point courant du polygône à clipper et sa position par rapport au côté clippant courant si c'est positif
   s_in = (cp20-cp10)*(s1-cp11) - (cp21-cp11)*(s0-cp10)
   #On calcule le même critère pour le deuxième coin du côté courant du polygône à clipper
   for e in inputList:
      e0=e[0]
      e1=e[1]
      e_in = (cp20-cp10)*(e1-cp11) - (cp21-cp11)*(e0-cp10)
      #Ainsi on sait que pour le côté ES du polygône à clipper si E et S sont du bon côté
      if e_in>=0:
      #E se trouve du bon côté il faudra donc le rajouter
         if s_in<0:
         #Cas ou le côté courant se trouve de part et d'autres du côté clippant on ajoute l'intersection
            outputList.append(computeIntersection_pair(cp10,cp11,cp20,cp21,s0,s1,e0,e1))
         outputList.append(e)
         #Cas ou S est du bon côté mais pas E
      elif s_in>=0:
          #Cas ou le côté courant se trouve de part et d'autres du côté clippant on ajoute l'intersection
         outputList.append(computeIntersection_pair(cp10,cp11,cp20,cp21,s0,s1,e0,e1))
         #Mais on ajoute pas S car il deviendra E à l'itération suivante
      #Le premier coin du polygône devient
      s0,s1,s_in = e0,e1,e_in
   #Si le polygone est déjà vide après avoir été clippé par un côté
   if outputList.size()<1:
       return outputList
   #On reinitialise le côté clippant comme commencant au deuxième point du côté précédent
   cp10, cp11 = cp20, cp21
return outputList
```

Annexe E

Liste des publications

E.1 Journaux Internationaux

— J. Ogier du Terrail, S. Agarwal, F. Jurie. *Recent Advances in Object Detection in the age of Convolutional Deep Neural Networks*, IJCV. Soumis, en cours d'évaluation.

E.2 Conférences internationales

— J. Ogier du Terrail, F. Jurie. *On the use of Deep Neural Networks for the detection of small vehicles in Aerial imagery*, IEEE International Conference on Image Processing 2017, Pékin, Chine.

E.3 Conférence nationale

— J. Ogier du Terrail, F. Jurie. *On the use of Deep Neural Networks for the detection of small vehicles in Aerial imagery*, Orasis 2017, Colleville sur mer, France. Oral.

Bibliographie

[1] Intelligence campus défi tech. `http://www.intelligencecampus.com/defis.html`.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems. *ArXiv e-prints*, March 2016.

[3] S. Agarwal, J. Ogier Du Terrail, and F. Jurie. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. *ArXiv e-prints*, September 2018.

[4] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11) :2189–2202, 2012.

[5] Anelia Angelova, Yaser Abu-Mostafam, and Pietro Perona. Pruning training sets for learning of object categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 494–501. IEEE, 2005.

[6] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS

'68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[7] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2911–2918. IEEE, 2012.

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv :1701.07875*, 2017.

[9] David Arthur and Sergei Vassilvitskii. K-means++ : The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[11] Lucas Beyer, Alexander Hermans, and Bastian Leibe. Biternion Nets - Continuous Head Pose Regression from Discrete Training Labels. In *GCPR*, pages 157–168, Cham, 2015. Springer, Cham.

[12] Mikołaj Bińkowski, Dougal J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv preprint arXiv :1801.01401*, 2018.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[14] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms – improving object detection with one line of code. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[15] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 7, 2017.

[16] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[17] Roberto Brunelli. *Template matching techniques in computer vision : theory and practice*. John Wiley & Sons, 2009.

[18] Adrian Bulat and Georgios Tzimiropoulos. Human Pose Estimation via Convolutional Part Heatmap Regression. In *ECCV*, pages 717–732, Cham, 2016. Springer International Publishing.

[19] Peter J Burt and Edward H Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics (TOG)*, 2(4) :217–236, 1983.

[20] Michal Busta, Lukas Neumann, and Jiri Matas. Deep textspotter : An end-to-end trainable scene text localization and recognition framework. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[21] Jennifer Carlet and Bernard Abayowa. Fast vehicle detection in aerial imagery. *arXiv preprint arXiv :1709.08666*, 2017.

[22] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4) :98, 2017.

[23] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet : An Information-Rich 3D Model Repository. Technical Report arXiv :1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[24] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote : synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16 :321–357, 2002.

[25] Nitesh V Chawla, Aleksandar Lazarevic, Lawrence O Hall, and Kevin W Bowyer. Smoteboost : Improving prediction of the minority class in boosting. In *European*

conference on principles of data mining and knowledge discovery, pages 107–119. Springer, 2003.

[26] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan : Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[27] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain adaptive faster R-CNN for object detection in the wild. *CoRR*, abs/1803.03243, 2018.

[28] G. Cheng, P. Zhou, and J. Han. RIFD-CNN : Rotation-Invariant and Fisher Discriminative Convolutional Neural Networks for Object Detection. In *CVPR*, 2016.

[29] Gong Cheng, Peicheng Zhou, and Junwei Han. Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(12) :7405–7415, 2016.

[30] Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip H. S. Torr. BING : Binarized normed gradients for objectness estimation at 300fps. In *IEEE CVPR*, 2014.

[31] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2414–2422. Curran Associates, Inc., 2016.

[32] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[33] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[34] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 137–145, New York, NY, USA, 1984. ACM.

[35] Timothy F Cootes, Gareth J Edwards, and Christopher J Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6) :681–685, 2001.

[36] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3) :273–297, 1995.

[37] James Coughlan, Alan Yuille, Camper English, and Dan Snow. Efficient deformable template detection and localization without user initialization. *Computer Vision and Image Understanding*, 78(3) :303–319, 2000.

[38] Gabriela Csurka. A comprehensive survey on domain adaptation for visual applications. In Gabriela Csurka, editor, *Domain Adaptation in Computer Vision Applications.*, Advances in Computer Vision and Pattern Recognition, pages 1–35. Springer, 2017.

[39] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment : Learning augmentation policies from data. *arXiv preprint arXiv :1805.09501*, 2018.

[40] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[41] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN : object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[42] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR, abs/1703.06211*, 1(2) :3, 2017.

[43] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[45] Zhipeng Deng, Hao Sun, Shilin Zhou, Juanping Zhao, and Huanxin Zou. Toward Fast and Accurate Vehicle Detection in Aerial Images Using Coupled Region-Based Convolutional Neural Networks. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10 :3652–3664, 2017.

[46] Emily L Denton, Soumith Chintala, Rob Fergus, et al. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

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

[48] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv :1703.04933*, 2017.

[49] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. Modeling visual context is key to augmenting object detection datasets. In *The European Conference on Computer Vision (ECCV)*, page 18, September 2018.

[50] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. On the importance of visual context for data augmentation in scene understanding. *arXiv preprint arXiv :1809.02492*, 2018.

[51] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn : Surprisingly easy synthesis for instance detection. In *The IEEE international conference on computer vision (ICCV)*, 2017.

[52] Ivan E. Sutherland and Gary W. Hodgman. Reentrant polygon clipping. *Commun. ACM*, 17 :32–42, Jan. 1974.

[53] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *iccv*, page 1033. IEEE, 1999.

[54] Mark Everingham, S. M. Eslami, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge : A retrospective. *Int. J. Comput. Vision*, 111(1) :98–136, January 2015.

[55] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal voc challenge. *International Journal of Computer Vision*, 88(2) :303–338, 2010.

[56] P. Felzenszwalb, R. Girshick, D. Mcallester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9) :1627–1645, 2009.

[57] Carles Fernández, Ivan Huerta, and Andrea Prati. A comparative evaluation of regression learning algorithms for facial age estimation. In *Face and Facial Expression Recognition from Real World Videos*, pages 133–144. Springer, 2015.

[58] Martin A. Fischler and Robert C. Bolles. Random sample consensus : A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6) :381–395, June 1981.

[59] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4340–4349, 2016.

[60] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation : Insights and applications. In *Deep Learning Workshop, ICML*, 2015.

[61] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv :1409.7495*, 2014.

[62] Michael S Gashler and Stephen C Ashmore. Training deep fourier neural networks to fit time-series data. In *International Conference on Intelligent Computing*, pages 48–55. Springer, 2014.

[63] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv :1705.07485*, 2017.

[64] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016.

[65] Dariu M Gavrila and Vasanth Philomin. Real-time object detection for" smart" vehicles. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 87–93. IEEE, 1999.

[66] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving ? the kitti vision benchmark suite. pages 3354–3361, 2012.

[67] Georgios Georgakis, Arsalan Mousavian, Alexander C. Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics : Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

[68] Spyros Gidaris and Nikos Komodakis. Locnet : Improving localization accuracy for object detection. *CoRR*, abs/1511.07763, 2015.

[69] Spyros Gidaris and Nikos Komodakis. Attend refine repeat : Active box proposal generation via in-out localization. In *BMVC*, 2016.

[70] Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[71] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[72] Clark R Givens, Rae Michael Shortt, et al. A class of wasserstein metrics for probability distributions. *The Michigan Mathematical Journal*, 31(2) :231–240, 1984.

[73] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS 2010). Society for Artificial Intelligence and Statistics*, 2010.

[74] Diego Marcos Gonzalez, Michele Volpi, Nikos Komodakis, and Devis Tuia. Rotation equivariant vector field networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[75] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[76] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv :1412.6071*, 2014.

[77] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[78] Guodong Guo, Yun Fu, Charles R Dyer, and Thomas S Huang. Head pose estimation - Classification or regression ? In *ICPR*, pages 1–4. IEEE, 2008.

[79] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2315–2324, 2016.

[80] Song Han, Huizi Mao, and William J Dally. Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv :1510.00149*, 2015.

[81] Kota Hara, Raviteja Vemulapalli, and Rama Chellappa. Designing deep convolutional neural networks for continuous object orientation estimation. *arXiv preprint arXiv :1702.01499*, 2017.

[82] Robert M Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67(5) :786–804, 1979.

[83] Robert M Haralick, Hyonam Joo, Chung-Nan Lee, Xinhua Zhuang, Vinay G Vaidya, and Man Bae Kim. Pose estimation from corresponding point data. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6) :1426–1446, 1989.

[84] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 447–456, 2015.

[85] Kevin Hartnett. To build truly intelligent machines teach them cause and effects, 2018. https ://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/.

[86] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn : Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[87] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[91] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv :1603.05027*, 2016.

[92] Tong He, Zhi Tian, Weilin Huang, Chunhua Shen, Yu Qiao, and Changming Sun. Single Shot TextSpotter with Explicit Alignment and Attention. *arXiv*, cs.CV, March 2018.

[93] David J Heeger and James R Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238. ACM, 1995.

[94] Geremy Heitz and Daphne Koller. Learning spatial context : Using stuff to find things. In *ECCV*. European Conference on Computer Vision, October 2008.

[95] João F. Henriques and Andrea Vedaldi. Warped convolutions : Efficient invariance to spatial transformations. In *ICML*, 2017.

[96] J.R. Hershey and Peder A. Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *IEEE ICASSP*, volume 4, pages IV–317–IV–320, 2007.

[97] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[98] Dang Ha The Hien. A guide to receptive field arithmetic for convolutional neural networks, 2017. https ://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807.

[99] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. *CoRR*, abs/1710.10710, 2017.

[100] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv :1207.0580*, 2012.

[101] Judy Hoffman, Sergio Guadarrama, Eric S Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda : Large scale detection through adaptation. In *Advances in Neural Information Processing Systems*, pages 3536–3544, 2014.

[102] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada : Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv :1711.03213*, 2017.

[103] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *European conference on computer vision*, pages 340–353. Springer, 2012.

[104] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257, 1991.

[105] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[106] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv :1709.01507*, 2017.

[107] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet : An efficient densenet using learned group convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[108] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[109] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[110] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv :1801.07365*, 2018.

[111] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[112] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks : Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1) :6869–6898, 2017.

[113] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet : Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv :1602.07360*, 2016.

[114] Naoto Inoue, Ryosuke Furuta, Toshihiko Yamasaki, and Kiyoharu Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. *CoRR*, abs/1803.11365, 2018.

[115] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv :1502.03167*, 2015.

[116] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[117] ISPRS. 2d semantic labeling contest, 2016. http ://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html.

[118] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[119] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv :1611.08207*, 2016.

[120] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe : Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[121] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang. Acquisition of Localization Confidence for Accurate Object Detection. *ArXiv e-prints*, July 2018.

[122] Alexia Jolicoeur-Martineau. The relativistic discriminator : a key element missing from standard gan. *arXiv preprint arXiv :1807.00734*, 2018.

[123] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[124] Andrej Karpathy. Cs231 : Convolutional neural networks for visual recognition, 2016. http ://cs231n.github.io/.

[125] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv :1710.10196*, 2017.

[126] Andras Kemeny. oktal, 1989. http ://www.oktal.fr/en/.

[127] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning : Generalization gap and sharp minima. *arXiv preprint arXiv :1609.04836*, 2016.

[128] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. Pvanet : deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv :1608.08021*, 2016.

[129] Davis E. King. Dlib-ml : A machine learning toolkit. *Journal of Machine Learning Research*, 10 :1755–1758, 2009.

[130] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2018.

[131] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

[132] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet : Towards accurate region proposal generation and joint object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 845–853, 2016.

[133] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages : A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https ://storage.googleapis.com/openimages/web/index.html*, 2017.

[134] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[135] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[136] Darius Lam, Richard Kuzma, Kevin McGee, Samuel Dooley, Michael Laielli, Matthew Klaric, Yaroslav Bulatov, and Brendan McCord. xview : Objects in context in overhead imagery. *CoRR*, abs/1802.07856, 2018.

[137] Agata Lapedriza, Hamed Pirsiavash, Zoya Bylinskii, and Antonio Torralba. Are all training examples equally valuable ? *arXiv preprint arXiv :1311.6510*, 2013.

[138] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv :1512.09300*, 2015.

[139] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[140] Yann LeCun, Koray Kavukcuoglu, Clément Farabet, et al. Convolutional networks and applications in vision. In *ISCAS*, volume 2010, pages 253–256, 2010.

[141] Jens Leitloff, Dominik Rosenbaum, Franz Kurz, Oliver Meynberg, and Peter Reinartz. An operational system for estimating road traffic information from aerial images. *Remote Sensing*, 6(11) :11315–11341, 2014.

[142] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv :1712.09913*, 2017.

[143] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco : Common objects in context. In *European Conference on Computer Vision (ECCV)*, Zürich, 2014. Oral.

[144] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[145] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv :1708.02002*, 2017.

[146] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017.

[147] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. *arXiv preprint arXiv :1804.07723*, 2018.

[148] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts : Differentiable architecture search. *arXiv preprint arXiv :1806.09055*, 2018.

[149] Kang Liu and Gellért Máttyus. Fast multiclass vehicle detection on aerial images. *IEEE Geosci. Remote Sensing Lett.*, 12 :1938–1942, 2015.

[150] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD : Single shot multibox detector. In *ECCV*, 2016.

[151] Yuliang Liu and Lianwen Jin. Deep matching prior network : Toward tighter multi-oriented text detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[152] Zikun Liu, Jingao Hu, Lubin Weng, and Yiping Yang. Rotated region based cnn for ship detection. In *Image Processing (ICIP), 2017 IEEE International Conference on*, pages 900–904. IEEE, 2017.

[153] Zikun Liu, Liu Yuan, Lubin Weng, and Yiping Yang. A high resolution optical satellite image dataset for ship recognition and some new baselines. In *ICPRAM*, pages 324–331, 2017.

[154] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110, 2004.

[155] Jianqi Ma, Weiyuan Shao, Hao Ye, Li Wang, Hong Wang, Yingbin Zheng, and Xiangyang Xue. Arbitrary-oriented scene text detection via rotation proposals. *IEEE Transactions on Multimedia*, 2018.

[156] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[157] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.

[158] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *International Conference on Computer Vision*, pages 89–96. IEEE, 2011.

[159] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2813–2821. IEEE, 2017.

[160] Francisco Massa, Bryan C Russell, and Mathieu Aubry. Deep exemplar 2d-3d detection by adapting from real to rendered views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6024–6033, 2016.

[161] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[162] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net : Fully convolutional neural networks for volumetric medical image segmentation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 565–571. IEEE, 2016.

[163] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv :1802.05957*, 2018.

[164] Michael Mol. Rosetta code, 2007. http ://www.rosettacode.org/wiki/.

[165] Kwang Moo Yi, Yannick Verdie, Pascal Fua, and Vincent Lepetit. Learning to assign orientations to feature points. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 107–116, 2016.

[166] Taylor Mordan, Nicolas Thome, Matthieu Cord, and Gilles Henaff. Deformable part-based fully convolutional network for object detection. *arXiv preprint arXiv :1707.06175*, 2017.

[167] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism : Going deeper into neural networks, 2015. https ://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[168] T. Nathan Mundhenk, Goran Konjevod, Wesam A. Sakla, and Kofi Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning. In *ECCV*. European Conference on Computer Vision, October 2016.

[169] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[170] Jean Ogier Du Terrail and Frédéric Jurie. On the use of deep neural networks for the detection of small vehicles in ortho-images. In *IEEE International Conference on Image Processing*, Beijing, China, September 2017.

[171] Junier B Oliva, Barnabás Póczos, and Jeff G Schneider. Distribution to Distribution Regression. *ICML*, 2013.

[172] Rupert Paget and I Dennis Longstaff. Texture synthesis via a noncausal nonparametric multiscale markov random field. *IEEE transactions on image processing*, 7(6) :925–931, 1998.

[173] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1) :15–33, Jun 2000.

[174] B. Peng, W. Tan, Z. Li, S. Zhang, D. Xie, and S. Pu. Extreme Network Compression via Filter Group Approximation. *ArXiv e-prints*, July 2018.

[175] Xingchao Peng, Baochen Sun, Karim Ali 0002, and Kate Saenko. Learning Deep Object Detectors from 3D Models. 2015.

[176] Bojan Pepik, Rodrigo Benenson, Tobias Ritschel, and Bernt Schiele. What is holding back convnets for detection ? In *German Conference on Pattern Recognition*, pages 517–528, 2015.

[177] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3) :313–318, 2003.

[178] Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[179] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv :1802.03268*, 2018.

[180] Kris Popat and Rosalind W Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Visual Communications and Image Processing'93*, volume 2094, pages 756–769. International Society for Optics and Photonics, 1993.

[181] Xiaojuan Qi, Qifeng Chen, Jiaya Jia, and Vladlen Koltun. Semi-parametric image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8808–8816, 2018.

[182] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv :1511.06434*, 2015.

[183] Anant Raj, Vinay P. Namboodiri, and Tinne Tuytelaars. Subspace alignment based domain adaptation for rcnn detector. In Xianghua Xie, Mark W. Jones, and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 166.1–166.11. BMVA Press, September 2015.

[184] Param S. Rajpura, Ravi S. Hegde, and Hristo Bojinov. Object detection using deep cnns trained on synthetic images. *CoRR*, abs/1706.06782, 2017.

[185] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[186] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net : Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[187] Sebastien Razakarivony. *Apprentissage de variétés pour la Détection et Reconnaissance de véhicules faiblement résolus en imagerie aerienne.* Theses, Université de Caen Basse-Normandie, December 2014.

[188] Sebastien Razakarivony and Frédéric Jurie. Discriminative autoencoders for small targets detection. In *IAPR international conference on pattern recognition*, pages 3528–3533, 2014.

[189] Sébastien Razakarivony and Frédéric Jurie. Vehicle Detection in Aerial Imagery : A small target detection benchmark. *Journal of Visual Communication and Image Representation*, December 2015.

[190] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf : an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[191] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv :1802.01548*, 2018.

[192] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[193] Joseph Redmon and Ali Farhadi. Yolo9000 : Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[194] Joseph Redmon and Ali Farhadi. Yolov3 : An incremental improvement. *arXiv preprint arXiv :1804.02767*, 2018.

[195] Jospeh Redmon. Darknet : Open source neural networks in c, 2015. https ://pjreddie.com/darknet/.

[196] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN : Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[197] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[198] Ton Roosendaal. Blender website, 1995. https ://www.blender.org/.

[199] Amir Rosenfeld, Richard Zemel, and John K Tsotsos. The elephant in the room. *arXiv preprint arXiv :1808.03305*, 2018.

[200] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut" : Interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 309–314, New York, NY, USA, 2004. ACM.

[201] Mohammad Amin Sadeghi and David Forsyth. 30hz object detection with dpm v5. In *European Conference on Computer Vision*, pages 65–79. Springer, 2014.

[202] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[203] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.

[204] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2 : Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[205] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

[206] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

[207] Bernt Schiele and James L Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, pages 610–619. Springer, 1996.

[208] Henry Schneiderman and Takeo Kanade. A statistical method for 3d object detection applied to faces and cars. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 746–751. IEEE, 2000.

[209] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet : A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[210] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat : Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[211] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[212] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod : Learning deeply supervised object detectors from scratch. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 3, page 7, 2017.

[213] Zhiqiang Shen, Honghui Shi, Rogério Schmidt Feris, Liangliang Cao, Shuicheng Yan, Ding Liu, Xinchao Wang, Xiangyang Xue, and Thomas S. Huang. Learning object detectors from scratch with gated recurrent feature pyramids. *CoRR*, abs/1712.00886, 2017.

[214] Baoguang Shi, Xiang Bai, and Serge Belongie. Detecting oriented text in natural images by linking segments. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[215] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016.

[216] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv :1703.00810*, 2017.

[217] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[218] Bharat Singh and Larry S. Davis. An analysis of scale invariance in object detection snip. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[219] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (12) :1349–1380, 2000.

[220] Lars W. Sommer, Tobias Schuchert, Jurgen Beyerer, Firooz A. Sadjadi, and Abhijit Mahalanobis. Deep learning based multi-category object detection in aerial images. In *SPIE Defense+ Security*, May 2017.

[221] Lars Wilko Sommer, Tobias Schuchert, and Jürgen Beyerer. Fast deep vehicle detection in aerial images. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 311–319. IEEE, 2017.

[222] Hang Su, Xiatian Zhu, and Shaogang Gong. Deep learning logo detection with data expansion by synthesising context. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 530–539. IEEE, 2017.

[223] Baochen Sun and Kate Saenko. From virtual to reality : Fast adaptation of virtual object detectors to real domains. In *BMVC*, volume 1, page 3, 2014.

[224] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1) :11–32, 1991.

[225] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[226] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[227] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[228] Kevin Tang, Vignesh Ramanathan, Li Fei-Fei, and Daphne Koller. Shifting weights : Adapting object detectors from image to video. In *Advances in Neural Information Processing Systems*, pages 638–646, 2012.

[229] Tianyu Tang, Shilin Zhou, Zhipeng Deng, Lin Lei, and Huanxin Zou. Arbitrary-Oriented Vehicle Detection in Aerial Imagery with Single Convolutional Neural Networks. *Remote Sensing*, 9 :1170–17, November 2017.

[230] Tianyu Tang, Shilin Zhou, Zhipeng Deng, Huanxin Zou, and Lin Lei. Vehicle detection in aerial images based on region convolutional neural networks and hard negative example mining. *Sensors*, 17(2), 2017.

[231] Franklin Tanner, Brian Colder, Craig Pullen, David Heagy, Michael Eppolito, Veronica Carlan, Carsten Oertel, and Phil Sallee. Overhead imagery research data set : An annotated data library & tools to aid in the development of computer vision algorithms. In *2009 IEEE Applied Imagery Pattern Recognition Workshop (AIPR 2009)*, pages 1–8. IEEE, 2009.

[232] thtrieu. Darkflow, 2015. https ://github.com/thtrieu/darkflow.

[233] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data : Bridging the reality gap by domain randomization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[234] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 4, 2017.

[235] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2) :154–171, 2013.

[236] Manik Varma and Debajyoti Ray. Learning the discriminative power-invariance trade-off. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[237] Gul Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[238] Andrea Vedaldi, Varun Gulshan, Manik Varma, and Andrew Zisserman. Multiple kernels for object detection. In *International Conference on Computer Vision*, pages 606–613. IEEE, 2009.

[239] Paul Viola and Michael J. Jones. Robust real time face detection. *International Journal of Computer Vision*, pages 137–154, 2004.

[240] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[241] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment : from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4) :600–612, 2004.

[242] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117. Eurographics Association, 2009.

[243] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans : A consistency term and its dual effect. *arXiv preprint arXiv :1803.01541*, 2018.

[244] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6) :343–349, June 1980.

[245] Bichen Wu, Forrest N Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet : Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *CVPR Workshops*, pages 446–454, 2017.

[246] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. Gp-gan : Towards realistic high-resolution image blending. *arXiv preprint arXiv :1703.07195*, 2017.

[247] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota : A large-scale dataset for object detection in aerial images. In *IEEE CVPR*, 2018.

[248] Yu Xiang, Feng Zhou, and Manmohan Chandraker. Deep Deformation Network for Object Landmark Localization. *arXiv*, May 2016.

[249] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns :

How to train 10,000-layer vanilla convolutional neural networks. *arXiv preprint arXiv :1806.05393*, 2018.

[250] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[251] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[252] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv :1505.00853*, 2015.

[253] Jiaolong Xu, Sebastian Ramos, David Vázquez, and Antonio M López. Domain adaptation of deformable part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(12) :2367–2380, 2014.

[254] Yoshihiro Yamada, Masakazu Iwamura, and Koichi Kise. Shakedrop regularization. *arXiv preprint arXiv :1802.02375*, 2018.

[255] T. Yang, X. Zhang, W. Zhang, and J. Sun. MetaAnchor : Learning to Detect Objects with Customized Anchors. *ArXiv e-prints*, July 2018.

[256] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[257] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[258] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv :1611.03530*, 2016.

[259] D. Zhang, J. Yang, D. Ye, and G. Hua. LQ-Nets : Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *ArXiv e-prints*, July 2018.

[260] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv :1805.08318*, 2018.

[261] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. Is faster r-cnn doing well for pedestrian detection ? In *European Conference on Computer Vision*, pages 443–457. Springer, 2016.

[262] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Schechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[263] Ziming Zhang, Jonathan Warrell, and Philip HS Torr. Proposal generation for object detection using cascaded ranking svms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1497–1504. IEEE, 2011.

[264] Qiyang Zhao, Zhibin Liu, and Baolin Yin. Cracking bing and beyond. In *BMVC*, 2014.

[265] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net : Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv :1606.06160*, 2016.

[266] Haigang Zhu, Xiaogang Chen, Weiqun Dai, Kun Fu, Qixiang Ye, and Jianbin Jiao. Orientation robust object detection in aerial images using deep convolutional neural network. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3735–3739. IEEE, 2015.

[267] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[268] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, pages 465–476, 2017.

[269] Larry Zitnick and Piotr Dollar. Edge boxes : Locating object proposals from edges. In *ECCV*. European Conference on Computer Vision, September 2014.

[270] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv :1707.07012*, 2(6), 2017.

Réseaux de Neurones Convolutionnels Profonds pour la détection de petits véhicules en Imagerie Aérienne

Résumé

Cette thèse présente une tentative d'approche du problème de la détection et discrimination des petits véhicules dans des images aériennes en vue verticale par l'utilisation de techniques issues de l'apprentissage profond ou "deep-learning". Le caractère spécifique du problème permet d'utiliser des techniques originales mettant à profit les invariances des automobiles et autres avions vus du ciel. Nous commencerons par une étude systématique des détecteurs dits "single-shot", pour ensuite analyser l'apport des systèmes à plusieurs étages de décision sur les performances de détection. Enfin nous essayerons de résoudre le problème de l'adaptation de domaine à travers la génération de données synthétiques toujours plus réalistes, et son utilisation dans l'apprentissage de ces détecteurs.

Mots clés : Apprentissage statistique ; détection d'objets ; apprentissage profond ;vision par ordinateur

Abstract

The following manuscript is an attempt to tackle the problem of small vehicles detection in vertical aerial imagery through the use of deep learning algorithms. The specificities of the matter allows the use of innovative techniques leveraging the invariance and self similarities of automobiles/planes vehicles seen from the sky. We will start by a thorough study of single shot detectors. Building on that we will examine the effect of adding multiple stages to the detection decision process. Finally we will try to come to grips with the domain adaptation problem in detection through the generation of better looking synthetic data and its use in the training process of these detectors.

Keywords : Statistical Learning ; object-detection ; deep-learning ; computer-vision