Mise en application d'intelligence artificielle pour éloigne les oiseaux

par

Stephane GU

RAPPORT DE PROJET PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE COMME EXIGENCE PARTIELLEÀ L'OBTENTION DE LA MAITRISE EN GÉNIE DE LA PRODUCTION AUTOMATISÉE
M. Ing.

MONTRÉAL, LE 22 AOÛT 2019

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Tony Wong, directeur de projet
Département de génie des systèmes à l'École de technologie supérieure

M. Mathieu Hamel, codirecteur de projet
CTO, Lockbird

M. Michel Rioux, évaluateur
Département de génie des systèmes à l'École de technologie supérieure

MISE EN APPLICATION D'INTELLIGENCE ARTIFICIELLE POUR ELOIGNER DES OISEAUX

STEPHANE GU

RÉSUMÉ

L'entreprise Lockbird propose une solution autonome d'effarouchement d'oiseaux aux sites d'enfouissements. Les oiseaux représentent un gros enjeu économique pour les professionnels de la gestion de déchet. Lockbird combine des solutions classiques d'effarouchement d'oiseaux, comme le laser ou des enregistrements audios et de l'intelligence artificielle pour la détection et localisation d'oiseaux sous forme de tourelle.

Comme l'entreprise veut utiliser un système embarqué, nous avons donc de fortes contraintes en puissance de calcul qui complique le projet. De plus, pour permettre un effarouchement efficace, nous avons aussi comme contrainte d'avoir un temps d'inférence inférieur à 5 secondes. La présence de déchet sur les images complexifie encore plus la détection.

Après la revue de littérature, nous avons donc décidé de travailler sur le modèle Faster R-CNN qui permet d'avoir des résultats en classification plus robuste. L'obstacle du projet a donc été de réduire le temps d'inférence de notre modèle pour satisfaire les attentes de l'entreprise sans trop affecter les performances en classification.

En effet, un moyen facile de réduire le temps d'inférence est de limiter le nombre de proposition de région que la couche de détection envoie. Cependant, pour descendre en dessous de 5 secondes en temps d'inférence, il faut un maximum de 100 propositions. Ce qui a des conséquences assez négatives sur la classification.

Pour remédier à ce problème nous avons donc optimiser les performances de la couche de propositions de région en optimisant différents hyperparamètres, comme la taille des ancres ou les poids de la régularisation.

Mots-clés : Intelligence artificielle, apprentissage automatique, détection d'objet, détection d'oiseaux.

BIRD DETERRENT USING APPLIED MACHINE LEARNING

STEPHANE GU

ABSTRACT

Lockbird is a company that offers an autonomous system for bird deterrent on landfill sites. Birds represent an economic challenge and safety hazard in waste management. Standard techniques are not effective once the fowls get used to the deterrent. A novel approach is thus needed for this task.

A promising solution combining turret-mounted lasers (or audio clips) guided by machine learning to detect and localize birds is being developed. On-site deployment calls for an embedded system with limited memory and computing power. In addition, to allow for efficient targeting, machine learning inference time for detection should be less than 5 seconds. Finally, the presence of waste and other objects in a landfill environment further complicates the detection procedure.

After reviewing the pertinent technical literature, we have decided on the Faster R-CNN architecture that allows for robust classification results. The main obstacle of the project was to constrain the inference time below a specified upper bound while maintaining classification performances.

An obvious way to reduce inference time was to limit the number of region proposals generated by the output detection layer. However, in order to have an inference time lower than 5 seconds, and with the current hardware configuration, the maximum number of region proposals is about 100. However, such a small sample size could have negative consequences on the classification performances.

To alleviate this problem, we optimized the region proposal layer by tuning multiple hyperparameters such as the anchor size or the regularization weights.

Key words: Artificial intelligence, machine learning, object detection, bird detection.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

RoI	Region of Interest
SVM	Support Vector Machine
RPN	Region Proposal network
R-CNN	Region Convolutional Neural Network
SPP-net	Spatial Pyramid Pooling network
FRCNN	Faster R-CNN
SSD	Single Shot Detector
cls	Classification
reg	Regression
API	Application Programming Interface
IOU	Intersection over Union
AR	Average recall
AP	Average precision
AIoU	Average Intersection over Union
AIT	Average Inference Time

INTRODUCTION

Dans le cadre de ce projet d'intervention en entreprise, nous avons travaillé au sein de Lockbird pendant une durée de 2 sessions. Notre rôle au sein de cette entreprise a été de travailler sur leur système de détection d'oiseaux en utilisant nos connaissances en détection d'objet. La détection d'objet a toujours été un problème difficile en vision par ordinateur mais avec beaucoup d'enjeux à la clé. En effet, de la sécurité à la conduite automatisée, développer un système de détection d'objet robuste et efficace pourrait révolutionner le monde, surtout depuis la découverte récente des réseaux de neurones par convolution.

Lockbird est une entreprise qui travaille avec les gestionnaires de sites d'enfouissements pour les aider à réduire la quantité d'oiseaux venant se nourrir dans les déchets. En effet, pour les sites d'enfouissements, les oiseaux venant se nourrir dans le site représente une perte non négligeable d'argent mais aussi un challenge pour l'environnement. Le principe de fonctionnement d'un site d'enfouissement est simple, il existe des zones de transbordements qui se chargent de réunir les déchets. Avoir des oiseaux dans la zone devient alors détrimentaire pour le site, mais aussi pour les oiseaux qui risquent de se faire écraser par les bulldozers chargés de compacter les ordures. Une fois les déchets réunis et compactés, ils sont ensuite transportés à la zone d'enfouissement ou ils seront enterrés. Les oiseaux deviennent encore plus contraignant dans cette zone, puisqu'ils endommagent le produit utilisé pour enterrer les déchets, forçant les entreprises de gestion de déchets à enterrer de nouveau les déchets ou à employer des moyens peu efficace mais aussi peu éthique assez souvent.

Lockbird a donc décidé de répondre à ce besoin des gestionnaires de sites d'enfouissement en combinant une solution écologique et éthique. Pour cela, l'entreprise utilise des systèmes traditionnels d'effarouchement d'oiseaux, comme un laser ou des enregistrements audio, combiné à une caméra intelligente qui utilise de l'intelligence artificielle pour détecter et localiser les oiseaux. La caméra intelligente permet ainsi d'avoir un système automatisé pour

l’effarouchement d’oiseaux. Une fois les oiseaux repérés, la camera envoie les positions nécessaires à la visée laser, ou tout simplement un signal sonore.

Ainsi, dans ce rapport nous allons détailler le raisonnement appliqué pour le développement de nos modèles de détection d’objet. Nous commencerons par détailler les attentes et contraintes du projet, avant de présenter le travail réalisé dans la littérature et enfin les résultats et performances de nos modèles.

CHAPITRE 1

CONTRAINTE DU PROJET

Avant d'aborder les architectures possibles pour la détection d'objet, nous allons commencer par définir les contraintes du projet. Nous pouvons définir trois catégories de contraintes, les performances désirées imposées par l 'entreprise, les contraintes sur les objets à détecter et enfin les contraintes sur le système.

1.1 Performances désirées

Commençons par aborder les contraintes des algorithmes. Nous cherchons à éloigner des oiseaux automatiquement, on peut déjà poser des contraintes de performances sur nos algorithmes. Nous travaillons avec des sites d'enfouissement qui ont l'habitude « d'accueillir » des milliers d'oiseaux par jour. L'entreprise a donc imposé que notre *recall* (ou recall rate = $\frac{VP}{VP+FN}$) soit d'au moins 80% pour s'assurer que notre système soit capable d'effaroucher la plupart. De plus, comme nous travaillons avec des oiseaux sauvages, notre système doit être assez réactif pour pouvoir effaroucher efficacement les oiseaux. L'entreprise veut donc un temps d'inférence inférieur ou égale à 5 secondes. Enfin, comme il y a une multitude de déchets sur les images du site, il faut que notre modèle soit en mesure d'identifier correctement les oiseaux, ou plus exactement de ne pas confondre des déchets pour des oiseaux. L'entreprise veut donc avoir un minimum de faux positif. Or dans le contexte du projet, nous identifions la capacité de notre modèle à identifier correctement les objets comme un calcul de précision ($\frac{VP}{VP+FP} \approx \frac{Oiseaux}{Objets}$). L'entreprise exige donc un taux en précision de 80%. Nous détaillerons ces différents taux dans la partie méthodologie.

1.2 Objets à détecter

Discutons maintenant des contraintes liées aux données du projet, soit les images. Nous utilisons actuellement une caméra fixe sur la tourelle, il faut donc capturer la zone à défendre

dans sa quasi-entièreté, ce qui nous donne des images à haute résolution, la taille standard étant de 3840×2160 pixels. Sur ces grandes images, un oiseau est généralement de la taille d'un carré de 40 pixels. La Figure 1.1 donne un exemple de la taille relative des oiseaux par rapport aux objets de la zone à défendre.

Figure 1.1 : Exemple d'images de la base d'image de Lockbird

Nous devons donc majoritairement détecter des objets de petites tailles. Nous reviendrons plus en détails sur ce point pendant la revue de littérature, mais il est important de retenir que nous travaillons avec de petits objets dans une grande image. Une autre contrainte sur les images est dans l'établissement de la base de données. En effet, comme nous travaillons avec des oiseaux sauvages dans leur milieu « naturel », il est important d'utiliser des images similaires pour l'entrainement.

Figure 1.2: Exemple d'images de la base Caltech-USCD Bird 200

La plupart des images sont du type de la Figure 1.2. L'image est centrée sur l'oiseau et est généralement de petite taille. Ces différences rendent l'utilisation de la plupart des bases de données d'oiseaux disponible sur Internet inutilisable. Nous avons dû établir notre propre base de données pour réaliser l'entrainement de notre modèle.

1.3 Contraintes du système

Enfin, il reste les contraintes physiques du projet. La plus grande contrainte physique est lié au fonctionnement du système. Comme nous l'avons mentionné précédemment, nous utilisons des tourelles qui sont installées sur les différents sites d'enfouissement, n'ayant pas forcément accès à Internet. Nous travaillons donc sur un système embarqué, actuellement un système utilisant un Minix, un mini PC fonctionnant avec Ubuntu, pour réaliser l'inférence des images capturées par la caméra fixe. Nous avons donc une capacité mémoire ainsi qu'une puissance de calcul limitée. De plus, comme nous installons les tourelles à l'extérieur, nous devons aussi nous soucier de la météo. Dans ce rapport, nous ne nous soucierons que des problèmes en lien avec les performances de nos modèles (par exemple, l'étanchéité du système ou la condensation sur la caméra). Nous nous intéressons plutôt à la luminosité et contraste de l'image, la présence de flocons de neige et/ou de pluie qui rajoute du bruit. Comme la différence de météo au Canada est très prononcé, pour pouvoir entrainer un modèle robuste à la météo, il faut que notre base de données comporte un nombre égale d'image provenant d'hiver (avec neige et peu de luminosité) que d'été (pas de neige, soleil et bien éclairé).

1.4 Conclusion

Ainsi, nous pouvons résumer les attentes de l'entreprise à ces trois points :

- Avoir un recall rate de 80% ou plus.
- Avoir un temps d'inférence de 5 secondes ou moins.
- Avoir une précision de 80% ou plus.

Ces attentes sont standards pour les algorithmes de détection d'objet, mais ce projet reste tout de même complexe notamment à cause des contraintes établies par l'entreprise. En effet,

Lockbird est une start-up assez jeune. Elle ne possède donc pas encore toutes les données nécessaires pour l'entrainement des modèles. De plus, l'entreprise souhaite vendre un système autonome. Elle nous contraint donc à travailler avec un système embarqué, ce qui limite fortement la puissance de calcul et la mémoire disponible. Nous présentons maintenant la revue de littérature pour constater ce qui est applicable au vue des contraintes établies.

CHAPITRE 2

REVUE DE LITTÉRATURE

Dans la littérature, nous avons surtout trouvé des articles traitant de la détection d'objets et peu sur la détection d'oiseaux en milieux naturels. Nous allons donc présenter les méthodes de détection d'objet dans ce document.

Il existe deux grandes catégories d'algorithmes pour la détection d'objets :

- Les méthodes basées sur les régions d'images
- Les méthodes régressives

2.1 Les méthodes basées sur les régions d'images

Ces méthodes se basent sur un fonctionnement à trois étapes, proche du fonctionnement du cerveau humain, en scannant l'image de manière générale et ainsi trouver les régions d'intérêts, en extraire les données intéressantes pour enfin les classifier. Ils existent une multitude d'exemple d'algorithmes (SPP-net, R-FCN, FPN ou Mask R-CNN) mais à cause des contraintes en mémoire et puissance de calcul, nous avons décidé de n'étudier que les méthodes fondamentales aux méthodes basées sur les régions, ces méthodes étant moins lourdes :

- Region Convolutional Neural Network (R-CNN)
- Fast R-CNN
- Faster R-CNN

2.1.1 La méthode R-CNN

Selon (R. Girshick, 2014) cette approche peut se résumer en 3 étapes : la génération de région, l'extraction de caractéristiques et la classification.

Pour la génération de région, l'algorithme réalise une recherche sélective (*selective research*) pour générer 2000 proposition de région pour chaque image. Cette recherche s'appuie sur un

regroupement de haut en bas (*bottom-up grouping*) permettant de trouver plus efficacement les boites candidates de tailles arbitraire tout en réduisant l'espace de recherche.

Comme le montre la Figure 2.1, on fixe chaque région à une taille fixe de 224×224 (*warped into a fixed resolution*) et on utilise 5 couches de convolution et 2 couches complétement connectées pour en extraire un vecteur de caractéristique de taille 4096. L'avantage d'utiliser un réseau de neurone à convolution permet d'obtenir une représentation robuste des caractéristiques pour chaque région.

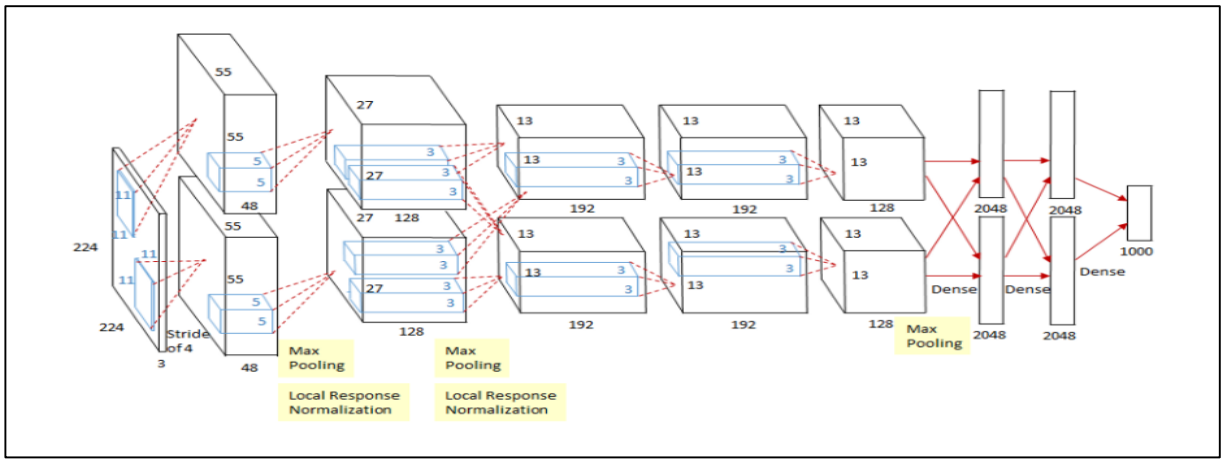


Figure 2.1: Réseau utilisé pour extraire les caractéristiques dans l'algorithm R-CNN

Enfin la classification se réalise à l'aide de Support Vector Machine (SVM) linéaires pour multiclasses. Ainsi, chaque proposition de région reçoit un score basé sur les régions positives et négatives (arrière-plan). Ce score permet d'ajuster les régions en réalisant une régression des cadres de sélections et on obtient les cadres de sélections finaux en filtrant avec suppression des non-maxima.

Cette méthode a apporté des améliorations aux méthodes traditionnelles, mais elle possède toujours des problèmes :

- La taille de l'image d'entrée doit être fixé. Ainsi, il faut toujours recalculer le réseau pour chaque région à évaluer entrainant ainsi des temps d'entrainement excessivement long. Ceci est dû à la présence de réseau complètement connecté (*fully connected layers*). Une alternative serait de forcer chaque région à avoir une taille fixe, mais cela peut entrainer des pertes d'information lié aux distorsions géométriques.

- L’entrainement de cette méthode se réalise en trois temps. Il faut entrainer le réseau de neurone à convolution, les classificateurs SVMs et enfin le regresseur de cadre de sélection. Ainsi, l’entrainement à un coût conséquent en mémoire et temps.
- Le temps pour obtenir les propositions de régions est non négligeable. Selon (R. Girshick, 2014), en utilisant une GPU Nvidia Tesla K20, le temps pour générer les propositions de régions est d’environ 13 secondes par images. Ce temps augmente à 53 secondes par images pour un CPU, le modèle du CPU n’étant pas spécifié. Ce problème est dû au fait que cette méthode réalise un forward pass pour chaque proposition de région.

Une partie des problèmes ont été résolu avec l’architecture Spatial Pyramid Pooling (SPP-net) dans (K. He, 2015), notamment le problème de taille fixe en utilisant des couches à convolution au lieu des couches complétement connectées et l’ajout d’une couche de spatial pyramid pooling. Ces couches de convolutions permettent aussi le partage des calculs, rendant ainsi le modèle R-CNN plus rapide. Mais le problème d’entrainement multiples reste un défi, et par conséquent le coût en espace mémoire. C’est ici qu’on introduit l’architecture Fast R-CNN.

2.1.2 La méthode Fast R-CNN

Selon (Girshick, 2015), cette méthode s’inspire de l’approche SPP-net et R-CNN. Cet algorithme prend en entrée une image entière et un ensemble de proposition d’objets. Le réseau parcourt l’image à l’aide de couches de convolutions et utilise ensuite des couches de max pooling pour créer une carte de caractéristiques de convolution. Pour chaque proposition d’objet, une couche de pooling par région d’intérêt (RoI) permet d’extraire un vecteur de caractéristique à taille fixe de la carte. Ces vecteurs parcourent ensuite une suite de couches complètement connecté (fc) qui finissent par deux couches de sorties, une qui permet de produire une estimation de probabilité softmax pour K classes d’objets et une classe « arrière-plan » et la seconde produit les coordonnées des cadres de sélection pour chaque classe d’objets. Ces coordonnées ont été affinés pour chaque classes.

Pour bien expliquer le fonctionnement de l'algorithme, nous allons revenir sur le fonctionnement en détails de la couche RoI (Region of Interest), ainsi que la méthode d'entrainement du modèle.

2.1.2.1 La couche de pooling RoI

Cette couche a pour but de convertir les caractéristiques d'une région d'intérêt en carte de caractéristiques de taille fixe. Ainsi, on fixe les hyper-paramètres H et W pour les dimensions de la carte. Dans l'article étudié, une région d'intérêt est une fenêtre rectangulaire défini par un tuple (*r,c,h,w*) tel que le coin en haut à gauche est défini par les coordonnées (*r,c*) et sa longueur et largeur par (*h,w*). Ainsi, la couche RoI prend une fenêtre de taille $w \times h$ puis crée une grille de taille $H \times W$ de sous fenêtre de taille $h/H \times w/W$. Les valeurs dans la grille sont obtenues par max pooling des sous-fenêtres associés à la case de la grille. Il est ainsi important d'appliquer cette couche de manière indépendante à chaque carte (*feature map channel*), comme dans un max pooling standard. On note que la couche RoI est un cas spécial de la couche de pooling par pyramide spacial utilisé dans SPP-net où il n'y a qu'un seul niveau à la pyramide.

2.1.2.2 Ajustement pour la détection

L'avantage de Fast R-CNN est qu'on peut entrainer tous les poids du réseau à l'aide de back propagation. En effet, la rétropropagation ne fonctionne pas très bien quand les échantillons d'entrainement, soient les régions d'intérêts, viennent d'images différentes (soit le fonctionnement de l'entrainement R-CNN et SPP-net). La rétropropagation ne fonctionne pas car chaque région d'intérêt à un grand champ récepteur, qui couvre souvent tout l'image d'entrée. Comme la propagation avant doit traiter tout le champ récepteur, les données d'entrée de l'entrainement sont donc souvent larges.

Pour la méthode Fast R-CNN, on propose une méthode qui utilise le partage de caractéristiques de l'entrainement. Pendant la descente de gradient stochastique, on échantillonne hiérarchiquement les mini-batch, en commençant par échantillonner *N* images,

puis en échantillonnant *R/N* régions d'intérêt par images. On profite du fait que pour les régions d'intérêt venant de la même image partagent les calculs et la mémoire pour la propagation avant et retro. Selon (S. Ren, 2015), l'idée semble ralentir la convergence de l'entrainement, vu que les régions d'intérêt d'une même image sont corrélées, mais en pratique cela ne semble pas être le cas.

2.1.2.3 Le processus d'entrainement

Comme nous l'avons indiqué plus haut, le réseau Fast R-CNN a deux couches de sorties sœurs. La première fournit une distribution de probabilité discrètes pour chaque région $p=(p_0,..,p_K)$ d'intérêt sur les $K+1$ classes. p est calculé avec un softmax sur les $K+1$ sorties d'une couche complétement connectée. La seconde couche fournit le décalage de la régression des cadres de sélection (*bounding-box regression offsets*), $t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$ pour chaque classe K indexé par k. t^k définie une translation indifférente à l'échelle et un décalage spatial relatif à une proposition d'objet (*log-space height/width shift relative to an object proposal*).

Contrairement aux méthodes précédentes, l'approche Fast R-CNN entraine ses couches de sorties simultanément. Pour cela, nous utilisons une fonction d'objectif agrégés L :

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1] L_{loc}(t^u, v) \quad (2.1)$$

Tiré de (Girshick, 2015) p.3

avec L_{cls} la perte de la première couche de sortie et L_{loc} la perte de la seconde.

Chaque région d'intérêt a comme label u la classe ground-truth et v la régression des cadres de sélection cibles (ground-truth aussi). Par définition, on pose $u = 0$ comme la classe arrière-plan. De plus, pour les régions d'intérêts de l'arrière-plan il n'y a pas de cadres de sélection ground-truth donc L_{loc} serait ignoré. $[u \geq 1]$ s'appelle l'opérateur d'Iverson et renvoie 1 quand $u \geq 1$ et 0 sinon.

Ainsi, $L_{cls}(p,u) = -log(p_u)$ est la perte logarithmique de la classe u. L_{loc} est défini en comparant $v = (v_x, v_y, v_w, v_h)$ le véritable tuple de la classe u et t^u le tuple prédit pour la classe u. La regréssion des bounding box est donc calculé de la manière suivante :

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^u - v_i) \tag{2.2}$$

Tiré de (Girshick, 2015) p.3

$$smooth_{L1} = \begin{cases} 0.5x^2 & si\ |x| \leq 1 \\ |x| - 0.5 & sinon \end{cases} \tag{2.3}$$

Tiré de (Girshick, 2015) p.3

$smooth_{L1}$ est une norme *L1* robuste choisie car moins sensible aux valeurs aberrantes (*outliers*) que la norme *L2* utilisé dans les algorithmes R-CNN et SPP-net. On introduit cette équation pour éviter les explosions de gradients lorsque les cibles de régressions sont sans bornes (*unbounded*).

2.1.2.4 La retropropagation avec les couches RoI

Le raisonnement employé pour réaliser une rétropropagation est la suivante, on note cependant que cette méthodologie sera expliquée pour une image par mini-batch, le passage à un nombre supérieur d'images étant trivial comme la propagation avant traite les images indépendamment :

Soit $x_i \in \mathbb{R}$ la *i*ème entrée d'activation de la couche RoI et soit y_{rj} *j*ème sortie de la couche *r*. Par définition, on a $y_{rj} = x_{i^*(r,j)}$ avec $i^*(r,j) = argmax_{i \in R(r,j)} x_i$ et R(r,j) est l'ensemble d'index des entrées de la sous fenêtre en train d'être évalué par y_{rj}. On note qu'un x_i peut être assigné à plusieurs y_{rj}.

Le calcul de dérivées partielles se réalise en fonction de chaque entrée x_i en suivant la formule suivante :

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{rj}} \tag{2.4}$$

Tiré de (Girshick, 2015) p.4

Tel que pour chaque mini-batch RoI *r* et pour chaque sortie de y_{rj}, la dérivée partielle $\frac{\partial L}{\partial y_{rj}}$ est accumulé si *i* est l'argmax sélectionné pour y_{rj} par max pooling.

Malgré ces améliorations dans l'architecture, la sélection des propositions de régions reste un goulot d'étranglement pour les calculs. En effet, on utilise le plus souvent des méthodes de recherches sélectives et Edgebox de (Dollar, 2014) pour générer l'ensemble des propositions de régions. Pour résoudre ce problème, nous introduisons la méthode Faster R-CNN.

2.1.3 La méthode Faster R-CNN

Comme l'explique (S. Ren, 2015), l'algorithme Faster R-CNN fonctionne avec deux modules. Le premier module est un réseau de neurones complètement connectés qui a pour but de proposer les régions et le second module est un détecteur Fast R-CNN utilisant les régions proposées par le 1er module. Comme nous avons détaillé le fonctionnement du détecteur dans la partie précédente, dans celle-ci nous ne présenterons donc que le module de proposition de régions nommé Region Proposal Network (RPN).

2.1.3.1 Le réseau de proposition de région

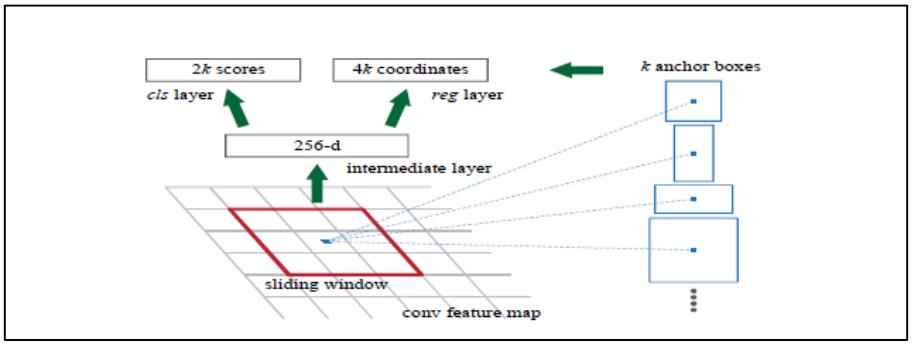


Figure 2.2: Architecture du RPN de Faster R-CNN

Tiré de (S. Ren, 2015) p.4

Comme le montre la Figure 2.2 tiré de (S. Ren, 2015), ce module a pour but de prendre comme entrée une image de taille quelconque et de proposer en sortie un ensemble de proposition d'objets rectangulaire avec un score associé (*objectness score*). Comme l'objectif est de partager les calculs avec le detecteur Fast R-CNN, les deux réseaux doivent partager un certain ensemble de couche de convolution (*conv layers*). Pour générer les propositions de régions, un petit réseau parcourt la carte de caractéristiques de convolution obtenu en sortie de la dernière couche de convolutions communes. Le réseau parcourant la carte prend comme entrée une fenêtre spatiale de dimension $n \times n$ et chaque fenêtre « coulissante » (*sliding window*) est mappé sur une dimension inférieure avant d'être transmise à deux réseaux complètements connectées sœurs, la couche de régression de cadres (*reg*) et la couche de classification de cadre (*cls*) de dimension 1×1. Il est intéressant de noter que comme ce mini-réseau fonctionne en « coulissant l'image » (*sliding window fashion*), les couches entièrement connectées sont partagées à travers toutes les positions spatiales (*spatial locations*).

Ainsi, pour chaque position de sous-fenêtre, on prédit simultanément plusieurs propositions de régions avec le maximum de possibilité noté k. Tel que la couche reg fournit $4k$ de coordonnées décrivant les k boites et la couche *cls* fournit $2k$ score d'estimation de probabilité pour savoir si c'est un objet ou non. Les k propositions sont paramétrés relativement aux k boites de références que l'on nomme ancres. Une ancre est centrée pour chaque sous-fenêtre et est associé à une échelle et un ratio de cadre (*scale and aspect ratio*) tel que si on a 3 échelles et 3 ratio de cadre, on aura 9 ancres pour chaque sous-fenêtre.

L'avantage de cette méthode est qu'elle développe une nouvelle approche pour la détection dans des images à échelles et ratios de cadres différentes. Comme illustrer dans la Figure 2.3, jusqu'à maintenant il existait 2 grandes approches pour gérer des images de tailles différentes. La première consistait à établir une pyramide d'images et de carte de caractéristiques associées. On modifie l'échelle de toutes les images pour pouvoir obtenir des caractéristiques en fonction de chaque échelle. Cette approche est évidemment très couteuse

en temps. La seconde approche vise à utiliser des noyaux de fenêtres coulissantes de taille différentes sur la carte des caractéristiques. On établit ainsi une « pyramide de filtre » en opposition à l'approche précédente.

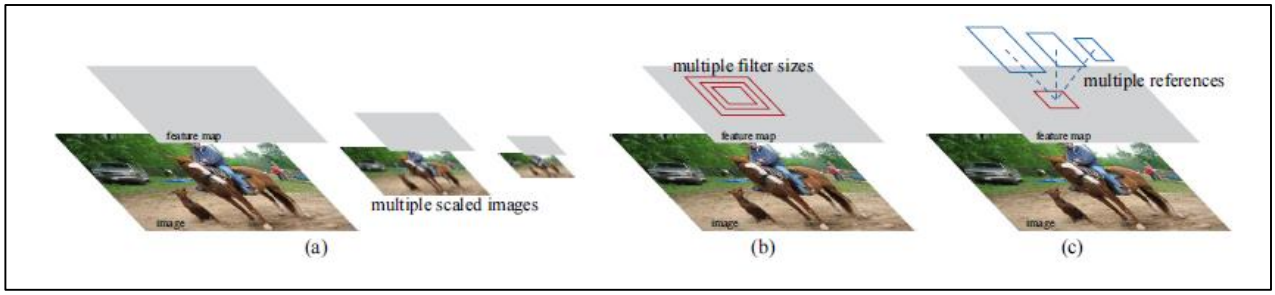


Figure 2.3: Les différentes méthodes pour gérer les images de tailles multiples. L'approche a) représente la pyramide d'images et de caractéristiques, b) la pyramide de filtres et c) la pyramide de boites de références

Tiré de (S. Ren, 2015) p.2

L'approche envisagée dans cet algorithme est de classifier et régresser les cadres de références en utilisant des ancres références de tailles différentes. Ainsi, on peut réaliser des études à échelle différentes à partir d'une échelle de taille unique pour l'image et les filtres. Cette approche permet ainsi de partager les caractéristiques calculées avec le détecteur sans devoir prendre en compte les différences d'échelles.

Considérons maintenant les fonctions d'objectifs de cet algorithme. Pour chaque ancre, on associe une classe label binaire. Le label positif est assigné pour deux types d'ancres :

1) L'ancre avec l'indice de Jaccard (*Intersection over Union*) du chevauchement avec le cadre ground-truth le plus élevé;

2) Une ancre avec un indice de Jaccard du chevauchement supérieur à 0.7 avec n'importe quelle cadre ground-truth.

Il est ainsi possible qu'un cadre ground-truth puisse assigner un score positif à plusieurs ancres. De manière générale, la seconde condition est suffisante, mais on adopte la première dans la situation où il n'y a pas d'échantillon positif. Le label négatif est assigné à une ancre négative quand l'indice de Jaccard est inférieur à 0.3 pour tous les cadres ground-truth. On note que les ancres neutres ne contribuent en rien à l'apprentissage.

Avec ces définitions on définit ainsi la fonction d'objectif pour une image, basé sur la fonction de Fast R-CNN :

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.5}$$

Tiré de (S. Ren, 2015) p.5

Avec *i* l'indice de l'ancre dans un mini-batch et p_i la probabilité prédite que l'ancre *i* soit un objet. p_i^* représente le ground-truth label et est égale à 1 si l'ancre est positive, 0 sinon. t_i conserve les 4 coordonnées du cadre de sélection prédit tandis que t_i^* conserve les coordonnées du cadre de sélection réel associé à une ancre positive. On définit aussi N_{cls} la taille du mini-batch et N_{reg} le nombre d'ancres ainsi que λ le facteur de régularisation. Enfin de la même manière que précédemment, L_{cls} représente la perte de classification et L_{reg} la perte de régression.

La fonction d'objectif L_{cls} est une perte logarithmique sur 2 classes (objet ou non). Pour la régression, on utilise $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ avec *R* la fonction $smooth_{L1}$ mentionnée plus tôt.

Le terme $p_i^* L_{reg}$ exprime le fait que la perte en régression (*regression loss*) ne s'active que pour des ancres positives. La sortie des couches *cls* et *reg* correspondent à $\{p_i\}$ et $\{t_i\}$ respectivement.

On normalise les deux termes avec N_{cls} et N_{reg} et on régularise avec λ. L'objectif de λ est de faire en sorte que la classification et la régression restent équilibrés, tel que par exemple si $N_{cls} = 200$ et $N_{reg} = 2000$, il faut $\lambda = 10$.

Pour la régression des cadres de sélections, on utilise les formules suivantes :

$$t_x = \frac{x - x_a}{w_a} \qquad t_y = \frac{y - y_a}{h_a}$$
$$t_w = \log(\frac{w}{w_a}) \qquad t_h = \log(\frac{w}{w_a}) \tag{2.6}$$
$$t_x^* = \frac{x^* - x_a}{w_a} \qquad t_y^* = \frac{y^* - y_a}{h_a}$$
$$t_w^* = \log(\frac{w^*}{w_a}) \qquad t_h^* = \log(\frac{w^*}{w_a})$$

Tiré de (S. Ren, 2015) p.5

Avec x, y, w et h les coordonnées du centre de la boite, sa largeur et hauteur. On a respectivement les variables x, x_a, et x^* pour la boite prédit, l'ancre et la boite ground-truth tel que la régression se réalise d'une ancre à une boite ground-truth proche, ce qui est différent des méthodes précédentes basées sur les régions d'intérêts. Cette différence est dû à la manière de gérer les différentes tailles de régions d'intérêts, les méthodes précédentes réalisant la régression sur des caractéristiques venant de région d'intérêts à taille arbitraire et les poids obtenus étant partagés pour toutes les tailles de régions. Dans cette méthode, on prend en compte les différences de tailles et on entraine un ensemble de k régresseurs qui ne partagent aucun poids, chacun étant utilisé pour une échelle et un format d'image. Le traitement est donc réalisé sur des cartes de caractéristiques de même taille, ce qui facilite le partage des caractéristiques avec le détecteur.

Cependant, nous n'avons considéré que l'entrainement du réseau de proposition de région sans considérer l'entrainement du détecteur Fast R-CNN qui va utiliser ces régions. Or, si nous entrainons le RPN et le détecteur Fast-RCNN indépendamment, alors les poids des couches de convolutions vont être modifier différemment. Il devient donc nécessaire de développer des couches de convolution partagées. Il existe pour cela 3 approches :

- L'entrainement alterné : On entraine d'abord le RPN puis on utilise les propositions établies pour entrainer le Fast R-CNN. Une fois le détecteur paramétré, on l'utilise comme entrée pour le réseau RPN et on réitère ce processus.
- L'entrainement joint approximatif : Pendant l'entrainement, les deux réseaux sont combinés en un. Pour chaque itération de la descente de gradient, la propagation avant génère des propositions de région qui sont traitées comme propositions fixes pour l'entrainement du détecteur Fast R-CNN. La rétropropagation se réalise normalement, et pour les couches partagées on combine les pertes des réseaux RPN et Fast R-CNN. Cette solution est facile à implémenter et plus rapide que la première méthode mais reste approximative car elle ignore la dérivée des coordonnées de cadres, qui sont aussi des sorties du réseau (*it ignores the derivative with respect to the proposal boxes' coordinates that are also network responses*).

- L'entrainement joint non approximatif : Comme indiqué plus haut, les cadres de sélection prédit par le RPN sont aussi des fonctions de l'entrée. Comme la couche RoI du détecteur accepte comme entrée les boites prédites, une bonne retropropagation devrait aussi prendre en compte les gradients des coordonnées des boites prédites.

Ainsi, avec l'ajout de cette méthode, il est devenu possible d'entrainer bout à bout les méthodes CNN basées sur les régions d'images mais il reste tout de même certains problèmes. En effet, la méthode alternative d'entrainement semble être trop couteuse en temps. De plus, la méthode RPN semble produire des régions qui ressemblent aux objets (arrière-plan inclus) au lieu d'instance d'objets. Cette méthode semble aussi avoir du mal à gérer les objets avec des formes extrêmes.

De plus, même si l'ajout de cette méthode permet d'entrainer le modèle en une seule étape, il est tout de même nécessaire de réalise un entrainement alterner des deux modules de l'algorithme ce qui est très couteux en temps. C'est pour remédier à ce problème que nous introduisons maintenant les méthodes régressives.

2.2 Les méthodes régressives

L'avantage des méthodes régressives est qu'elles réalisent la détection en une seule étape, en mappant directement les pixels des images en coordonnées de cadre de sélection et probabilité de classes, réduisant ainsi le temps de calcul. Pour la présentation de ces méthodes, nous présenterons les deux algorithmes les plus connus, YOLO (*You Only Look Once*) et SSD (*Single Shot Detector*).

2.2.1 You Only Look Once (YOLO)

Comme l'explique (Joseph Redmon, 2016), la détection des méthodes régressives se réalise en une étape. Ceci est dû au fait qu'elles n'utilisent qu'un réseau pour réaliser la détection d'objet. C'est notamment le cas de l'algorithme YOLO qui utilise les caractéristiques de l'image complète pour prédire chaque cadre de sélection.

Le principe de l'algorithme YOLO est de diviser l'image d'entrée en une grille SxS et que chaque case de la grille soit responsable de prédire l'objet centré dans la case. Chaque case prédit *B* cadres de sélection et leur score que l'on définit par $Pr(Objet) * IoU_{pred}^{truth}$ qui indique quelle est la probabilité que l'objet existe et à quel point on est confiant envers cette prédiction. Il faut cependant aussi prédire dans chaque case, et peu importe le nombre de boites, *C* probabilités conditionnelles de classes $Pr(Classe_i|objet)$. Ces probabilités et scores sont bien évidemment calculés que pour les cases ayant un objet tel que s'il n'y a pas d'objet le score de confiance est de 0 et s'il existe un objet, le score de confiance est l'indice de Jaccard (IOU) entre la boite prédite et la boite ground-truth.

Pendant la phase de test, on obtient les scores de confiance de classe en multipliant les prédictions de boites individuelles avec les probabilités conditionnelles de classes :

$$P(Objet) * IoU_{pred}^{truth} * P(Classe_i|objet) = P(Classe_i) * IoU_{pred}^{truth} \quad (2.7)$$

Tiré de (Joseph Redmon, 2016) p.2

tel qu'on obtient des scores de confiance pour les classes dans chaque boite. Ainsi, ces scores nous permettent de quantifier la probabilité qu'une classe apparaisse dans l'image ainsi qu'à quelle point la boite prédit corresponde à l'objet.

Selon la Figure 2.4, l'architecture de ce modèle comporte 24 couches de convolutions suivit de 2 couches entièrement connectés. Elle est inspirée par l'architecture GoogLeNet pour la classification d'image, à la différence que les modules inceptions sont remplacés par une couche de réduction 1x1 suivit d'une couche de convolution 3x3.

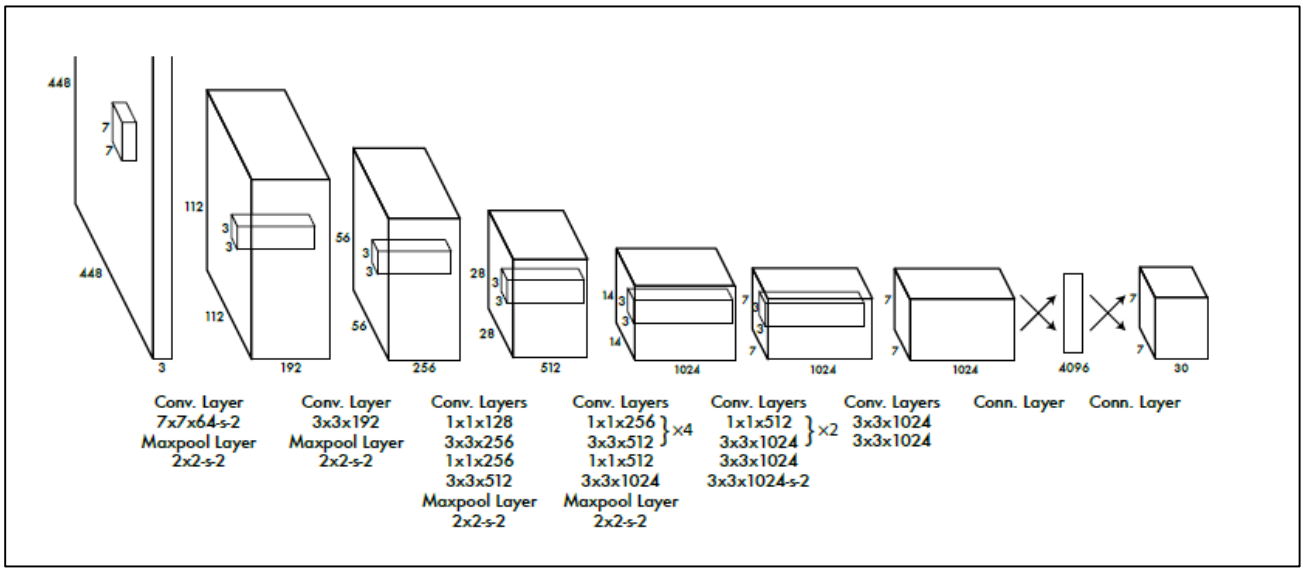


Figure 2.4: Architecture de l'algorithme YOLO
Tiré de (Joseph Redmon, 2016) p.3

On note que pour réaliser la détection, qui requiert souvent des informations visuelles précises, on fixe la résolution de l'image d'entrée à 448x448. Pour l'entrainement, on veut que chaque prédicteur de cadre de sélection soit responsable d'un objet. On assigne un prédicteur à un objet basé sur quelle prédiction a l'indice de Jaccard le plus grand avec le label. Ainsi, chaque prédicteur s'améliore dans la prédiction d'une taille, format d'image ou classe d'objet. On utilise la fonction d'objectif suivante pour l'optimisation :

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x_\iota})^2+(y_i-\hat{y_\iota})^2] \tag{2.8}$$

$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[\left(\sqrt{w_i}-\sqrt{\widehat{w_\iota}}\right)^2+\left(\sqrt{h_i}-\sqrt{\widehat{h_\iota}}\right)^2]$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left(C_i-\widehat{C_\iota}\right)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}\left(C_i-\widehat{C_\iota}\right)^2$$

$$+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\widehat{p_\iota}(c))^2$$

Tiré de (Joseph Redmon, 2016) p.4

tel que dans une case i, (x_i, y_i) décrit le centre du cadre prédit relativement à la case de la grille alors que ($\hat{x}_i, \hat{y}_i$) décrit les positions ground-truth de la boite. De la même manière, (w_i,h_i) représente la largeur et hauteur normalisée à la taille de l'image et ($\hat{w}_i, \hat{h}_i$) représente la vraie largeur et hauteur normalisée à la taille de l'image. C_i représente le score de confidence alors que $\hat{C}_i$ représente l'indice de Jaccard entre la boite prédite et celle ground-truth. $\mathbb{1}_i^{obj}$ indique l'existence d'un objet tandis que $\mathbb{1}_{ij}^{obj}$ indique que la prédiction a été réalisé par le $j$ème prédicteur de cadre de sélection. Enfin, λ_{coord} et λ_{noobj} sont des coefficient de régularisation des fonctions d'objectifs. L'idée dans (Joseph Redmon, 2016) est d'augmenter l'influence de la perte lié à la prédiction des coordonnées des bounding box et diminuer l'influence de la perte lié à la prédiction des boites ne contenant pas d'objet. Ainsi, dans l'article étudié, on fixe $\lambda_{coord} = 5$ et $\lambda_{noobj} = 0.5$.

Ainsi cet algorithme a théoriquement les avantages d'être extrêmement rapide et permettant une utilisation en temps réelle. De plus, comme l'algorithme utilise toute l'image pendant l'entrainement et les tests, le modèle prend en compte implicitement des informations contextuelles sur la classe en plus de leur apparence.

Il existe cependant des limitations fortes avec cet algorithme. En effet, comme cet algorithme impose des contraintes spatiales fortes, chaque case de la grille ne prédit que deux cadres et ne peut avoir qu'une classe. Ces contraintes limitent ainsi le nombre d'objets à proximité que le modèle peut détecter. Il a notamment du mal avec les petits objets en groupe, comme une volée d'oiseaux. De plus, la fonction d'objectif ne prend pas en compte la taille des cadres en cas d'erreur, ce qui est problématique puisqu'une erreur dans un petit cadre est plus dramatique que dans un grand cadre. Pour finir, cet algorithme a des problèmes de généralisation, surtout pour les objets avec un nouveau ratio hauteur/largeur (aspect ratio) et produit des caractéristiques grossières à cause de multiples opérations de sous-échantillonnage (downsampling). C'est pour résoudre ces problèmes que l'algorithme Single Shot MultiBox Detector a été inventé.

2.2.2 Single Shot Detector (SSD)

Selon (Wei Liu, 2016), cet algorithme s'inspire de l'approche RPN en reprenant l'utilisation des ancres. Ainsi, au lieu d'établir un grille fixe comme dans l'approche YOLO, le SSD utilise un ensemble d'ancres de défaut avec ratio de cadre et échelle différentes pour discrétiser l'espace de sortie des cadres de sélection. La différence avec ces approches est qu'on ne rééchantillonne pas les pixels ou caractéristiques (*pixel or feature resampling*).
Pour rentrer plus en détails, l'approche est basée sur une propagation avant d'un réseau de convolution qui produit un ensemble de cadres de sélection à taille fixe et de score pour la présence d'objet dans ces cadres.

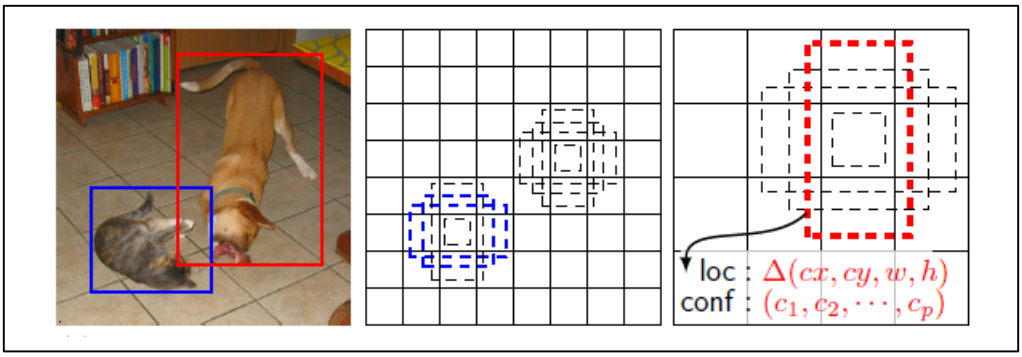


Figure 2.5: SSD framework
Tiré de (Wei Liu, 2016) p.3

Dans la Figure 2.5, nous avons un exemple avec à gauche l'image d'entrée avec les boites ground-truth pour chaque objet de l'entrainement. On établit alors l'ensemble des cadres de sélection par défaut de rapport de cadre différent pour chaque position de plusieurs cartes de caractéristiques avec différentes échelles (*default boxes of different aspect ratios at each location in several feature maps with different scales*), ici 8x8 et 4x4 pour l'image du milieu et celle de droite respectivement. Pour chaque boite par défaut, on prédit à la fois le décalage de forme (*shape offset*) et la confiance pour toutes les catégories d'objets.

Une fois ces cadres obtenus, on effectue une suppression des non maxima pour obtenir la détection finale. L'architecture de cette approche se concentre sur les couches de classification puisqu'elle utilise ce qu'on appelle un réseau de base, qui une architecture

standard utilisé pour de la classification d'images haute qualité mais sans aucune couche de classification. Nous allons ainsi présenter les couches de détection qui les remplacent.

Pour la détection, cette approche utilise des cartes de caractéristiques à échelle multiples, c'est le même principe que l'approche YOLO qui utilise une échelle unique. On utilise ainsi des couches de convolutions à la fin du réseau de base et ces couches diminuent progressivement de tailles et permettent ainsi la prédiction à échelles multiples. Comme nous pouvons le voir sur la Figure 2.6 chaque couche rajoutée sur le réseau de base permet d'avoir un ensemble fixe de prédiction en utilisant un ensemble de filtres à convolution.

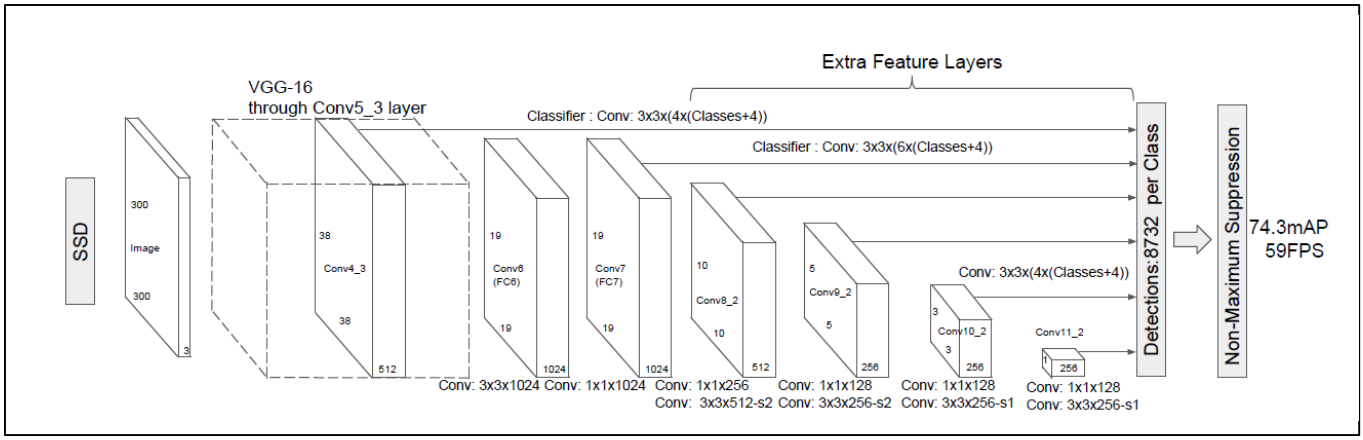


Figure 2.6: Architecture de l'approche SSD
Tiré de (Wei Liu, 2016) p.4

Ainsi, pour une couche de taille $m \times n$ avec p canaux, l'élément de base pour la prédiction de paramètres d'une potentielle détection est un petit noyau de dimension $3 \times 3 \times p$ qui produit soit un score pour la classe, ou un décalage de forme relatif aux coordonnées de la boite par défaut, tel qu'on ait $m \times n$ valeurs de sortie possibles, pour chaque position où on applique le noyau. Pour le décalage de cadre de sélection, les valeurs sont mesurées par rapport à la position des boites par défaut de chaque position de la carte de caractéristiques (*relative to a default box position relative to each feature map location*).

Revenons maintenant sur le principe de boite par défaut. Ces boites par défaut sont similaires aux ancres de l'approche Faster R-CNN, avec la seule différence qu'on les applique à

plusieurs cartes de caractéristiques de résolutions différentes. L'algorithme fonctionne de la manière suivante :
On associe un ensemble de cadres de sélection par défaut à chaque case de la carte de caractéristiques, pour plusieurs cartes au bout du réseau (*at the top of the network*). Ces cadres par défaut parcourt la carte de caractéristique par convolution tel que la position de chaque boite par rapport à sa case soit fixe. Ainsi, pour chaque grille de caractéristiques, on peut prédire le décalage par rapport à la forme du cadre par défaut dans la case mais aussi le score par classe qui indique la présence d'un élément de la classe à chaque case. Plus précisément, pour chaque boite parmi k, on calcule c scores de classes et les 4 décalages de la forme de la boite de base. On obtient ainsi un total de $(c+4)k$ filtres qui sont appliqué à chaque position dans la carte, on a alors $(c+4)kmn$ sorties pour une carte $m \times n$.

Abordons maintenant la méthodologie de l'entrainement. La différence principale avec l'approche SSD est que les informations ground-truth doivent être assignées à des sorties de détecteurs spécifiques avant de pouvoir réaliser la rétroprogation et l'optimisation. Il faut aussi choisir l'ensemble de boites par défaut, l'échelle pour la détection ainsi que l'équilibrage des données négative.

En effet, il faut choisir quelle boite par défaut corresponds à quelle détection ground-truth et entrainer le réseau en conséquence. Pour chaque boite ground-truth, on choisit des boites par défaut qui varie en position, rapport de cadre et échelle. Pour cela, on commence par associer chaque boite ground-truth à une boite par défaut avec le meilleur indice de Jaccard. On associe ensuite les boites par défaut à toutes les boites ground-truth ayant un indice de Jaccard supérieur à 0.5. Cela permet au réseau de prédire des scores élevés pour plusieurs boites par défaut se chevauchant plutôt que de ne choisir que celle qui a l'indice le plus élevé. Le choix d'échelles et de rapports hauteur/largeur des boites est nécessaire pour pouvoir détecter des objets de différentes tailles. Comme dans l'exemple de la Figure 2.5, en combinant les prédictions de différentes échelles on peut détecter le chien qui n'apparait que pour le noyau 8x8 et est considéré comme négatif par le noyau 4x4. Nous avons déjà abordé les différentes méthodes applicables pour résoudre ce problème et la méthode appliquée dans

SSD s'inspire de la pyramide d'images. La différence est dans la méthode d'obtention des différentes cartes de caractéristiques. Au lieu de modifier la taille de l'image, on récupère les cartes aux différentes couches du réseau. Ainsi, on a sensiblement le même résultat tout en conservant le partage de paramètres pour toutes les dimensions d'objets. Par conséquent, l'approche SSD utilise les cartes de caractéristiques des couches du début et celle de fin pour obtenir des cartes de tailles différentes (représenté par les cartes 8x8 et 4x4 de la Figure 2.5). Selon (Bolei Zhou, 2015), cette approche devrait poser problème vis-à-vis de la taille des champs réceptifs (*receptive field*). En effet, les cartes de caractéristiques de différentes couches d'un même réseau ont des tailles de champs récepteurs différentes. Mais cela n'affecte pas SSD car les boites par défaut ne doivent pas forcément correspondre au champ récepteur de chaque couche. En effet, le quadrillage des boites par défaut (*tiling of default box*) est conçu pour que les cartes de caractéristiques réagissent aux différentes tailles d'objets. Par exemple, imaginons que nous utilisons *m* cartes de caractéristiques pour la prédiction. L'échelle des boites par défaut pour chaque carte est calculée de la manière suivante :

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1) \ , \qquad k \in [1, m] \tag{2.9}$$

Tiré de (Wei Liu, 2016) p.6

Où s_{min} est 0.2 et s_{max} est 0.9 selon (Wei Liu, 2016). Ainsi, la couche la plus basse a une échelle de 0.2, la plus haute une échelle de 0.9 et les couches intermédiaires sont écartées de manière régulière. On impose plusieurs rapport de cadres pour les boites par défaut et les notes comme $a_r \in \{1,2,3,\frac{1}{2},\frac{1}{3}\}$. On peut ainsi calculer la largeur et hauteur pour chaque boite ($w_k^a = s_k\sqrt{a_r}$ $h_k^a = \frac{s_k}{\sqrt{a_r}}$). Pour le rapport de cadre de 1, on ajoute aussi une boite par défaut dont l'échelle est $s'_k = \sqrt{s_k s_{k+1}}$, tel qu'on a 6 boites par défaut pour chaque position de carte de caractéristiques. On définit aussi le centre de chaque boite par défaut par $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$ où $|fk|$ est la taille de la *k*ème carte de caractéristiques carré (*square feature map*) et *i*, *j* appartenant à $[0,|f_k|]$.

Avant d'aborder l'équilibrage des données négatives, il est nécessaire d'introduire la fonction d'objectif utilisé dans SSD. Elle est inspirée de l'approche MultiBox mais est modifié pour gérer plusieurs catégories d'objets. Soit $x_{ij}^{p} = \{1,0\}$ l'indicateur d'assignement de la *i*ème boite par défaut à la *j*ème boite ground-truth de catégorie *p*. Selon la méthode d'assignement établie plus haut, il est possible d'avoir $\sum_i x_{ij}^{p} \geq 1$. La fonction d'objectif utilisée est une somme pondérée de la perte en localisation (loc) et en confiance (conf) :

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g) \tag{2.10}$$

Tiré de (Wei Liu, 2016) p.5

Où N est le nombre de boites par défaut assignée. Si N = 0, alors on suppose que la perte est nulle. La perte en localisation est un perte Smooth_{L1} entre la boite prédit (l) et les paramètres de la boite ground-truth (g). Tout comme dans Faster R-CNN, on régresse les décalages du centre (cx,cy) de la boite par défaut (d) et pour sa largeur (w) et hauteur (h) tel qu'on calcule $\hat{g}$ la régression de la manière suivante :

$$L_{loc} = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,g)} x_{ij}^{k} smooth_{L1}(l_i^m \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = \frac{(g_j^{cx} - d_i^{cx})}{d_i^w} \qquad \hat{g}_j^{cy} = \frac{(g_j^{cy} - d_i^{cy})}{d_i^h} \tag{2.11}$$

$$\hat{g}_j^{w} = \log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^{h} = \log(\frac{g_j^h}{d_i^h})$$

Tiré de (Wei Liu, 2016) p.5

La perte de confiance (*confidence loss*) est comme un softmax sur plusieurs classes (c) et le poids α égale 1 par validation croisée :

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \ where \ \hat{c}_i^P = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{2.12}$$

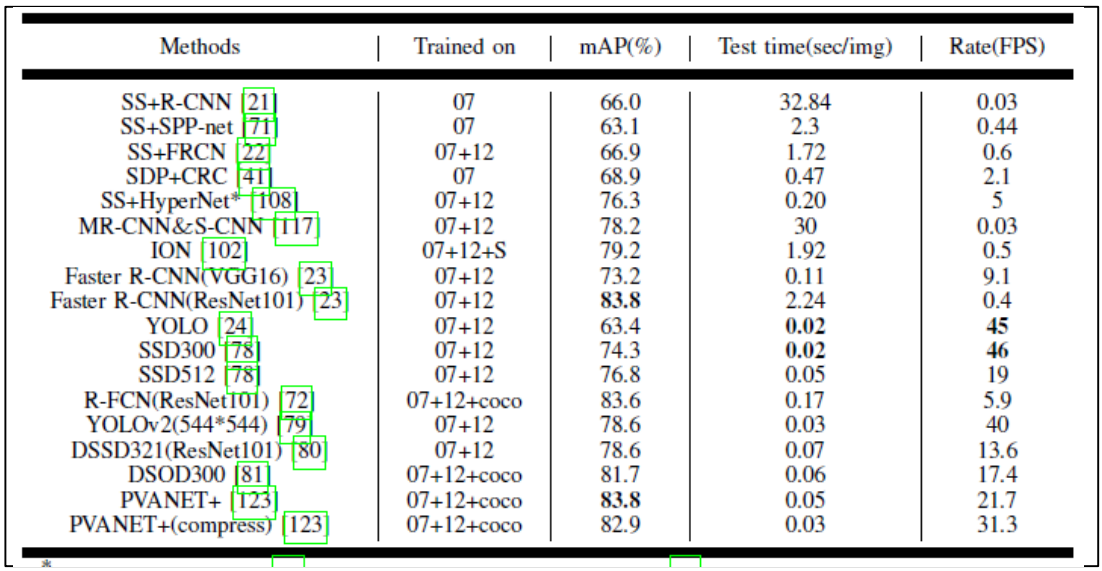
Tiré de (Wei Liu, 2016) p.5

Enfin, après l'étape d'assignement, il y a une grande majorité de boites par défaut négative, surtout si le nombre possible de boite par défaut est grand. Il y a donc déséquilibre entre les données positives et négatives. Pour y remédier, on trie les exemples négatifs par le score de confiance pour chaque boite par défaut et on ne conserve que ceux qui ont le loss en confiance le plus grand tel que le rapport entre les données négatives et positives est au plus 3 pour 1.

Ainsi, toutes ces étapes pendant l'entrainement permettent au modèle SSD d'obtenir de bon résultats en précision tout en étant extrêmement rapide, d'autant plus que la résolution de l'image est basse. Cependant, selon (C. Y. Fu, 2017) SSD semble toujours avoir des problèmes pour détecter les objets de petites tailles, ce qui pourrait être modifiable en utilisant de meilleur réseau de base, servant surtout comment extracteur de caractéristiques, ou en introduisant des couches de déconvolution.

2.2.3 Analyse critique

Maintenant que nous avons présenté les principales méthodes de détection d'objets, comparons leur performance. Nous ne comparons cependant que 2 des 3 critères que nous impose l'entreprise, la rapidité et la précision. En effet, pour les études comparatives trouvées, il n'y a pas de mention du recall. Nous n'avons donc évalué que la précision avec la précision moyenne (mean Average Precision) et la rapidité en images par seconde ou en frame par seconde (fps).
Ainsi, dans (Zhong-Qiu Zhao et al., July 2018), on obtient les résultats comparatifs suivant :



Methods	Trained on	mAP(%)	Test time(sec/img)	Rate(FPS)
SS+R-CNN [21]	07	66.0	32.84	0.03
SS+SPP-net [71]	07	63.1	2.3	0.44
SS+FRCN [22]	07+12	66.9	1.72	0.6
SDP+CRC [41]	07	68.9	0.47	2.1
SS+HyperNet* [108]	07+12	76.3	0.20	5
MR-CNN&S-CNN [117]	07+12	78.2	30	0.03
ION [102]	07+12+S	79.2	1.92	0.5
Faster R-CNN(VGG16) [23]	07+12	73.2	0.11	9.1
Faster R-CNN(ResNet101) [23]	07+12	**83.8**	2.24	0.4
YOLO [24]	07+12	63.4	**0.02**	**45**
SSD300 [78]	07+12	74.3	**0.02**	**46**
SSD512 [78]	07+12	76.8	0.05	19
R-FCN(ResNet101) [72]	07+12+coco	83.6	0.17	5.9
YOLOv2(544*544) [79]	07+12	78.6	0.03	40
DSSD321(ResNet101) [80]	07+12	78.6	0.07	13.6
DSOD300 [81]	07+12+coco	81.7	0.06	17.4
PVANET+ [123]	07+12+coco	**83.8**	0.05	21.7
PVANET+(compress) [123]	07+12+coco	82.9	0.03	31.3

Figure 2.7: Résultats de l'étude comparative des méthodes de détection d'objets

Tiré de (Zhong-Qiu Zhao et al., July 2018) p.11

Ces résultats ont été obtenus avec une CPU Intel i7-6700K et une GPU Nvidia Titan X et à partir de la base de test VOC07. Cette base comporte 9963 images contenant 24640 annotations réparties sur 20 classes comportant des animaux, des objets ainsi que des véhicules. La segmentation des données est généralement la suivante, 50% des images pour la base d'entrainement et de validation et 50% pour la base de test.

Selon la Figure 2.7, on définit les méthodes suivantes même si elles ne nous intéressent pas vraiment pour cette revue :

- SS : Selective Search
- SS* : fast mode Selective Search
- HyperNet* : version accéléré de HyperNet
- Pavnet+(compress) : Pavnet avec des votes de cadres de sélection et des couches entièrement connectés compressées
- SSD300/SSD512 : SSD avec la taille d'image d'entrées de 300x300 et 512x512 respectivement

On constate que le modèle avec la meilleure précision est la méthode Faster R-CNN alors que les méthodes SSD et YOLO sont bien plus rapides. Cela respecte les indications théoriques des articles étudiés.

Cependant, ce tableau ne prend pas en compte certains points importants de notre étude.
Pour cela, nous étudions l'étude comparative de (Jonathan Huang et al., 2017), commençons par comparer les algorithmes selon la vitesse et la précision.

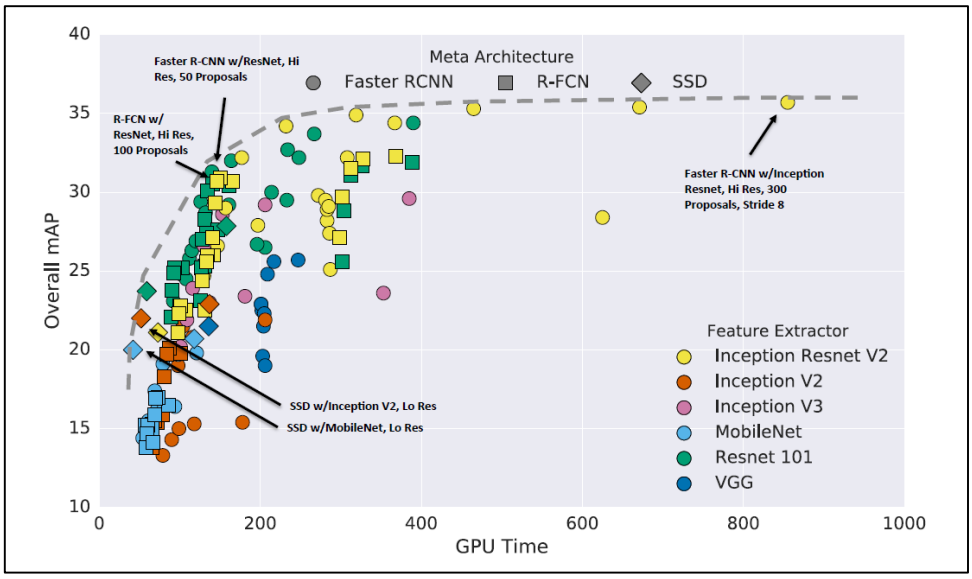


Figure 2.8: Précision vs vitesse
Tiré de (Jonathan Huang et al., 2017) p.6

La Figure 2.8 confirme ce que l'on trouve dans l'étude précédente, de manière générale la méthode SSD est plus rapide que la méthode Faster R-CNN, cependant cette dernière peut avoir de bien meilleur résultats en précision. Ainsi, pour pouvoir faire un choix entre ces 2 approches, il serait préférable d'observer plus en détails les performances.

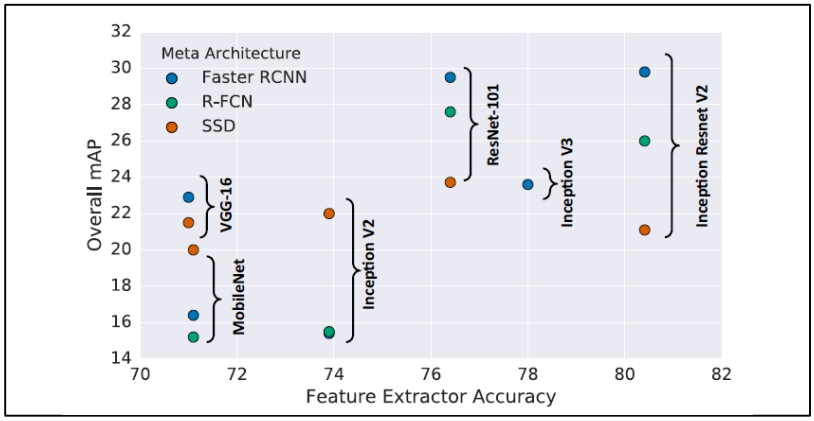


Figure 2.9 : Résultats de performance en fonction de l'architecture
Tiré de (Jonathan Huang et al., 2017) p.7

Dans la Figure 2.9, on compare les performances des méthodes Faster R-CNN et SSD en fonction de l'extracteur de caractéristiques choisit. On fixe pour cette étude la résolution de l'image d'entrée à 300. Cette courbe a pour but d'identifier l'impact des réseaux de base utilisé pour les différentes approches. On constate ainsi que l'approche SSD semble peu affecté par l'extracteur de caractéristiques alors que l'approche Faster R-CNN est très fortement impacté, tel que plus le réseau de base est efficace pour la classification plus elle mène à de meilleur résultats pour la détection d'objets. Ainsi, dans le cas où nous choisissons l'approche Faster R-CNN, il serait préférable d'utiliser Inception ResNet V2 comme extracteur de caractéristique pour maximiser la précision.

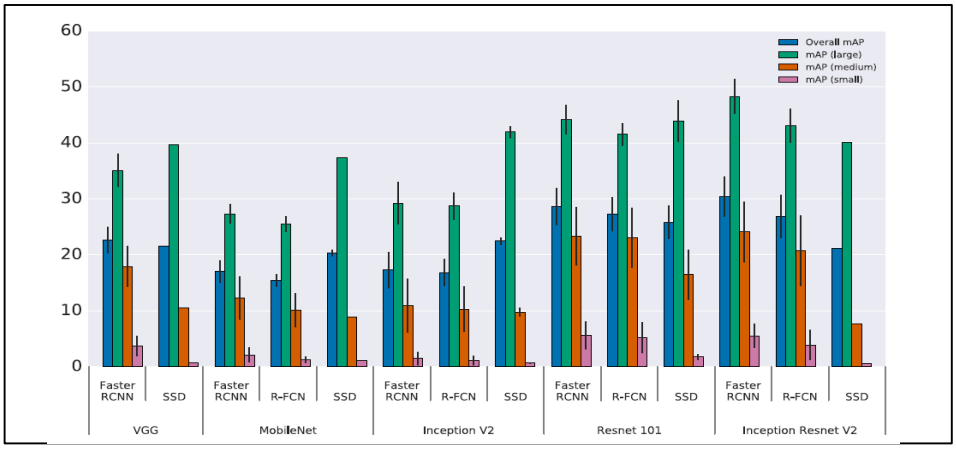


Figure 2.10 : Impact de la taille de l'objet sur la précision
Tiré de (Jonathan Huang et al., 2017) p.7

La Figure 2.10 représente l'impact de la taille de l'objet sur la précision. De manière intuitive, plus l'objet est large plus la détection se réalise bien. Cependant comme nous travaillons avec des objets de petites tailles, nous constatons qu'il serait préférable de choisir l'approche Faster R-CNN pour avoir des résultats satisfaisants.

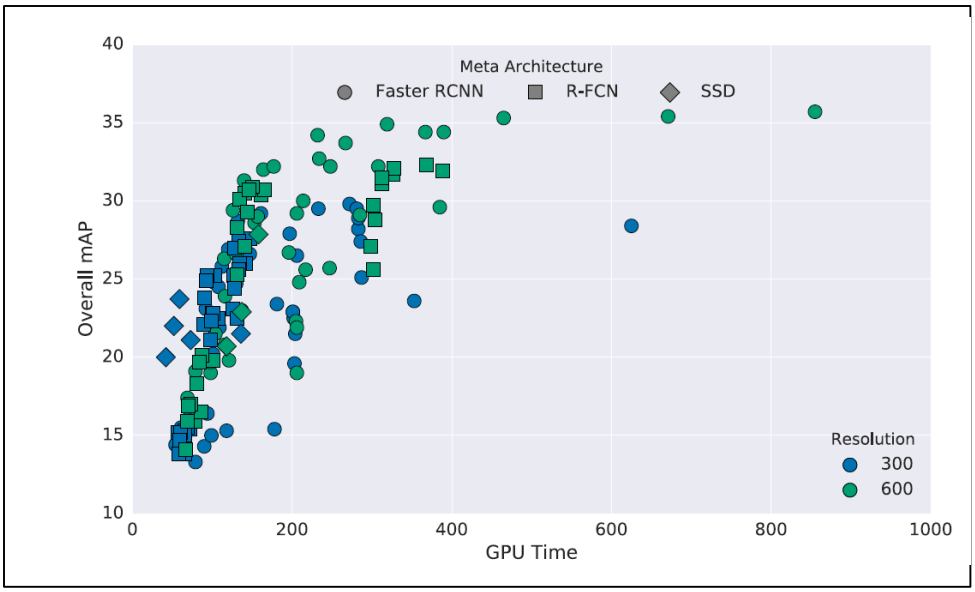


Figure 2.11 : Impact de la résolution de l'image sur les performances
Tiré de (Jonathan Huang et al., 2017) p.6

On constate selon la Figure 2.11 que la résolution de l'image a un impact conséquent sur les performances. Le cadre d'étude a été de tester l'impact d'une réduction de dimension de l'image. La conclusion est que les performances chutent de manière générale. Une autre conclusion qu'on peut tirer de cette courbe est que de manière générale, ces deux modèles ont de meilleurs résultats pour de petits objets quand on utilise des images à haute résolution. Ainsi, même si cette courbe ne nous permet pas de tirer de conclusion sur quel modèle choisir, elle reste intéressante vis-à-vis de notre contexte de projet.

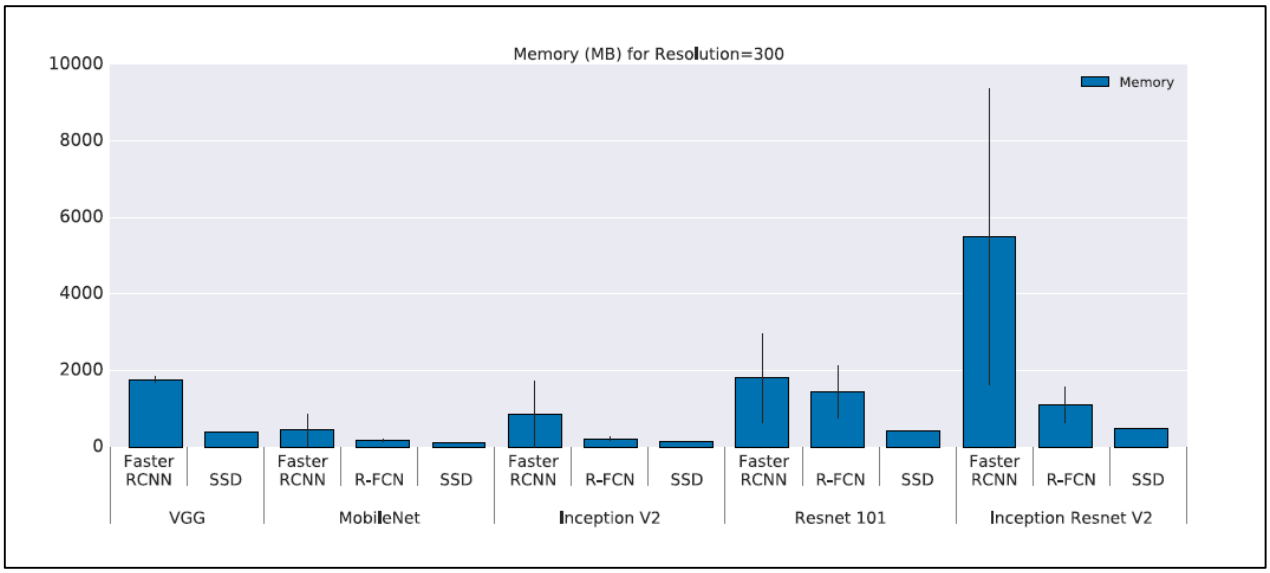


Figure 2.12 : Consommation mémoire des algorithmes
Tiré de (Jonathan Huang et al., 2017) p.7

Un autre point important de notre projet est l'utilisation mémoire qu'entraine ces algorithmes. En effet, comme nous travaillons avec des systèmes embarqués, il y a une forte contrainte sur l'espace mémoire utilisable. Selon la Figure 2.12, sans surprise, plus le modèle est lourd, plus il l'utilise de mémoire. Ainsi pour satisfaire la contrainte de mémoire il serait préférable de choisir l'architecture SSD. Cette courbe nous apporte plus d'informations sur l'architecture du modèle de base à utiliser. En effet, en consommation mémoire, si on venait à utiliser le modèle Faster R-CNN, il serait préférable d'utiliser l'architecture MobileNet pour la contrainte en mémoire, cependant les performances en précision sur cette architecture sont vraiment faibles. Le bon compromis entre mémoire et performances serait donc l'architecture Inception V2 pour Faster R-CNN. Le choix d'utiliser l'approche SSD facilite le choix, puisque l'impact du modèle de base sur les performances sont négligeables, la contrainte en mémoire prenant ainsi le dessus.

Enfin, nous concluons notre étude analytique des résultats par une courbe qui étudie l'impact du nombre de proposition de région sur l'architecture Faster R-CNN. Cette étude montre que

nous pouvons réduire le nombre de proposition de région sans impacter les performances de manière significatives. Ainsi, on constate avec cette courbe qu'il est possible d'accélérer l'approche Faster R-CNN en limitant le nombre de proposition de région à un maximum de 100. Selon la Figure 2.13, la courbe pleine nous indique les performances en précision des modèles tandis que la courbe à pointillé nous indique le temps d'inférence. On constate ainsi que le nombre idéal est surement 50 propositions de région, car on conserve 96% de la précision du modèle Inception Resnet v2 avec 300 propositions pour un temps de calcul diviser par 3.

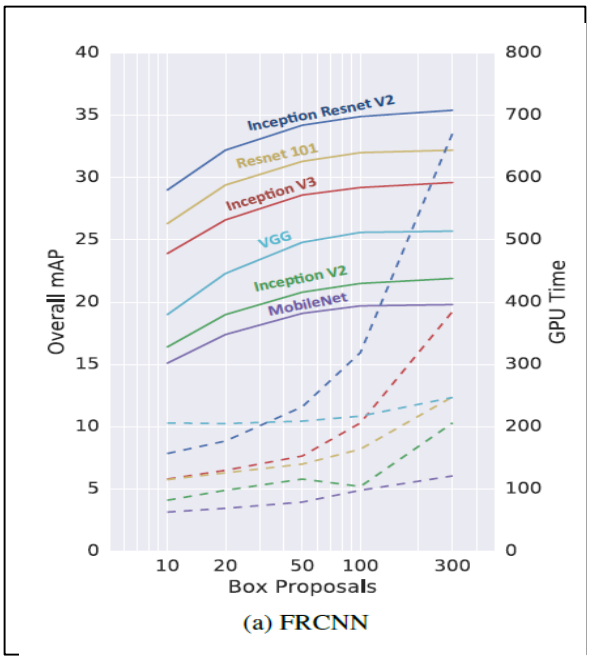


Figure 2.13 : Impact du nombre de proposition de région
Tiré de (Jonathan Huang et al., 2017) p.7

2.3 Conclusion

En conclusion, nous n'avons malheureusement pas de choix apparent d'approche. En effet, l'approche SSD satisfait les contraintes en mémoire et en rapidité, mais les difficultés de gérer les petits objets impactent trop les performances pour en faire un choix idéal. De la même manière, l'approche Faster R-CNN semble satisfaire les contraintes en précision, ainsi

qu’en temps si on limite le nombre de propositions de sujet mais au détriment de la consommation en mémoire.

Pendant la première partie du projet, l’entreprise a voulu prioriser les performances en temps plutôt qu’en précision/recall. Les machines étant déjà déployer sur des sites avec beaucoup d’oiseaux, Lockbird a voulu prioriser des modèles réactifs pour pouvoir les éloigner. Nous avons donc priorisé l’entrainement du modèle Mobilenet SSD V2 COCO. En effet, comme nous l’avons vu selon la revue de littérature, l’architecture Mobilenet a de bon résultats pour la plus petite consommation mémoire des autres architectures. Nous avons dans la suite cherché à améliorer les performances en précision de nos modèles SSD. Malheureusement, nous n’avons pas réussi à atteindre les performances attendues avec l’architecture SSD. Dans la deuxième partie du projet, nous avons donc décidé de travailler sur le modèle Faster R-CNN Inception V2. L’architecture Inception V2 a été choisi de la même manière que pour l’architecture Mobilenet. Nous avons priorisé un modèle léger sans pour autant trop sacrifier en précision. Nous détaillerons le raisonnement dans le chapitre suivant, avec la présentation des résultats obtenus pour le modèle Mobilenet SSD et pour le modèle Faster R-CNN.

CHAPITRE 3

METHODOLOGIE EXPERIMENTALE

Comme nous l'avons mentionné plus tôt dans les contraintes en données, il existe peu d'images d'oiseaux adaptées à l'utilisation voulue. N'ayant que des images venant de la base entreprise, nous avons jugé judicieux de réaliser du *transfer learning*, qui satisfait parfaitement nos attentes. Pour rappel, le *transfer learning* est le principe d'utiliser un modèle entrainé par d'autres chercheurs et de spécialiser ce modèle à notre utilisation souhaitée en entrainant la dernière couche du réseau avec nos données. Nous pouvons ainsi profiter des entrainements réalisés avec des bases de données beaucoup plus conséquentes que ce que nous possédons, mais cela nous permet aussi d'utiliser une quantité d'information bien plus petite que ce que nous aurions dû réunir. Pour pouvoir utiliser des modèles pré-entrainés, nous avons donc décidé de travailler avec Tensorflow object detection API (*Application Programming Interface*) et le langage Python. Pour utiliser l'API de tensorflow object detection, il faut obtenir les codes à partir du GitHub (https://github.com/tensorflow/models/tree/master/research/object_detection). Ces codes permettent de réaliser l'entrainement d'un modèle de détection d'objet à partir d'un modèle pré-entrainé. Pour faire fonctionner ces codes, il faut les ressources suivantes :

- Les données sous forme de TFrecords : Nous avons des données sous forme d'images avec annotations, enregistrer dans un fichier csv. L'API de tensorflow utilise cependant des TFRecords. Nous avons donc réalisé une fonction permettant la création des TFRecords.
- Le fichier de configuration : C'est dans ce fichier que l'on configure notre modèle. Ainsi, on y retrouve les différents hyperpamètres de nos modèles ainsi que les paramètres d'entrainements. Nous détaillerons plus en détails dans la partie résultat les changements et choix réalisé.

La Figure 3.1 détaille le déroulement de l'entrainement d'un modèle.

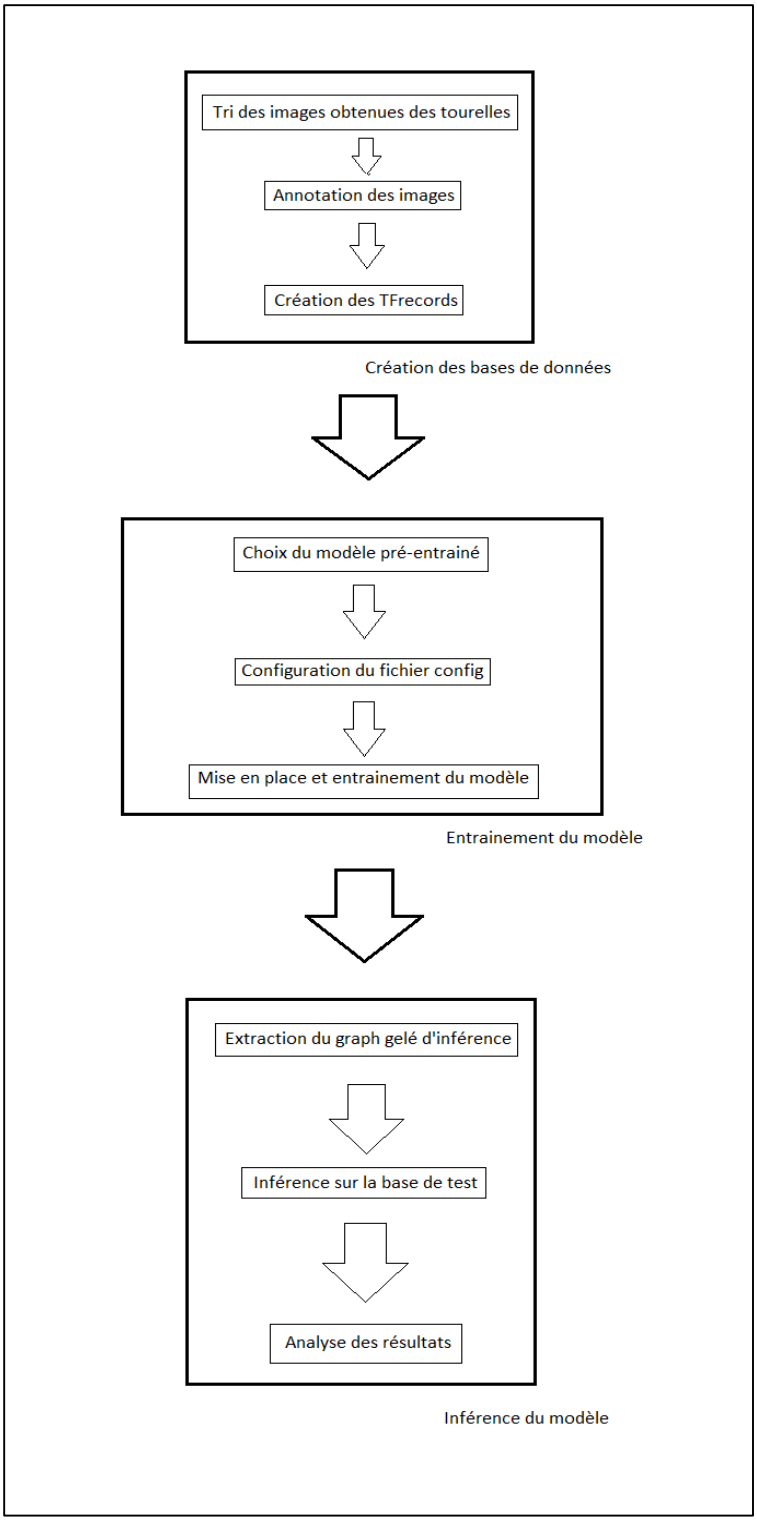


Figure 3.1 : Organigramme détaillant les étapes pour l'entrainement d'un modèle

Ainsi, nous allons détailler la méthodologie sur deux de ces variables du projet. Pour le modèle pré-entrainé, nous avons choisi les modèles Faster R-CNN Inception V2 et SSD Mobilenet v2 pour des raisons que nous avons détaillées dans la revue de littérature.

3.1 La création des jeux de données

Pour les données d'entrainements, nous avions initialement travaillé avec des images obtenues par l'entreprise pour nous familiariser avec l'API de Tensorflow. Ces images cependant ne représentaient pas l'utilisation que nous avions besoin (voir Figure 3.2).

Figure 3.2: Exemple d'images de la base initiale

La plupart de ces images représentent des oiseaux en vol ou au sol. Cependant, la plupart des oiseaux présents sur les sites d'enfouissements et que nous cherchons à éloigner sont au sol à se nourrir dans les déchets. Pour obtenir les données nécessaires à l'entrainement d'un modèle, l'entreprise met généralement en place deux ou trois tourelles sur un site d'enfouissement, et chaque tourelle envoie une image toutes les 20 secondes. Une fois ces données disponibles sur le drive, il fallait encore trier ces images et séparer les images comportant des oiseaux des images n'en comportant pas. Une fois ce tri réalisé, il ne restait plus qu'à les annoter pour pouvoir les utiliser. Il est important de noter que pour la plupart du projet, nous n'avions des images disponibles que pour le site de Waterloo, ainsi pour la suite du rapport, les images utilisées et les résultats obtenus ne seront que pour le site de Waterloo. À titre d'exemple, la Figure 3.3 montre l'environnement du site de Waterloo en période estivale et en période hivernale.

Pour réaliser les annotations, nous avons utilisé le logiciel LabelIMG (https://github.com/tzutalin/labelImg). Ce logiciel OpenSource permet la création d'annotation au format Pascal VOC. Nous ne détaillerons pas l'utilisation du logiciel, qui est extrêmement intuitif mais il est intéressant de noter que le logiciel renvoie un fichier xml avec toutes les informations nécessaires à la définition d'une bounding box, qu'il faut ensuite formater à un .csv que notre API peut utiliser, avant de créer les TFRecords avec. L'étape de formatage des xml en csv est relativement simple, mais il est intéressant de noter notre format pour les fichiers csv, chaque ligne se décompose de la manière suivante :

nom_image,largeur_image,epaisseur_image,classe,xmin,ymin,xmax,ymax

On note aussi que pour l'entrainement des modèles, nous n'utilisons qu'une seule classe « bird » pour identifier les oiseaux. La largeur et hauteur des images sont aussi fixes à 3840 et 2160 respectivement. Comme il fallait créer une base de données entière, nous avons pu diriger une équipe de 4 personnes et superviser la qualité des annotations créer. La tâche était assez complexe, compte tenu de la taille des images et la taille relative des oiseaux, ainsi que la présence de déchet pouvant prêter à confusion.

Figure 3.3 : Exemple d'images de Waterloo Winter (haut) et Summer (bas)

La méthodologie de la création de base de données fut simplifiée par le travail réalisé au préalable dans l'entreprise. En effet, pour respecter la partition des données d'entrainement en trois ensemble d'images (entrainement, validation et test) que l'on retrouve souvent dans la littérature, l'entreprise a établi un code permettant la séparation des données en ces trois sets. Ce code à l'origine séparait les annotations des fichiers csv, en trois fichiers csv différents. La segmentation était la suivante, le set d'entrainement regroupait 80% des données tandis que le set de validation et de test regroupait 10% respectivement. En utilisant les informations du fichier csv, il est alors possible de séparer les images en fonction du split réalisé sur les annotations. Nous avons cependant soulevé deux problèmes qui rendent la création d'un jeu de données plus complexe.

Tout d'abord, ce code est orienté pour la création d'image à partir d'une tourelle. Cependant, pour le site de Waterloo, nous travaillions avec deux tourelles. Comme le code parcours l'intégralité des csv pour pouvoir répartir en 3 ensembles, concaténer les fichiers csv de différentes tourelles ne permettait pas une répartition équitable des données. Le plus souvent le set d'entrainement était constitué des images de la première et deuxième tourelle tandis que le set de validation et test ne contenait des images que de la dernière tourelle.

Enfin, comme la séparation des sets se réalisaient en comptant le nombre de ligne total, il pouvait arriver qu'une image apparaisse à la fois dans le set d'entrainement et dans le set de validation par exemple, mais pas avec toutes les annotations. En effet, si une image apparaissait dans deux sets différents, il n'y avait toutes les annotations de l'image dans aucun des ensembles. Il fallait donc corriger ce bug puisque cela pouvait fausser les données d'entrainements.

Une fois l'automatisation de la création de base de données réalisées, il a été plus simple de créer des bases de données pour nos modèles à partir des images provenant des deux tourelles. Nous avons donc réalisé plusieurs bases de données que nous avons utilisé au cours du projet :

- La base PreparedData : base de données initialement utilisées, nous avons nettoyer une partie des données que nous considérions comme ayant trop de bruits. Elle possède 8000 annotations.
- La base Winter : les images de Waterloo_2 (la deuxième tourelle) avec des images d'hiver et la présence d'oiseaux au sol. La plus grande distinction de cette base de données vis-à-vis de Summer, est la présence de neige. Elle possède 10000 annotations.
- La base Summer : les images de Waterloo_1 et Waterloo_2 avec des images d'été. On note une baisse du nombre d'oiseaux constaté vis-à-vis de Winter. Cette différence est apparemment dû au fait que lorsque les images étaient prises, les mouettes étaient dans une autre région pour se reproduire. Elle possède 5000 annotations.
- La base Winter+Summer : On a ajouté aux images d'hiver des images d'été. Pour maintenir un équilibre entre les données d'hiver et d'été, nous avons ajouté que 5000 annotations de la base Winter.

Maintenant que nous établit les différentes bases de données que nous avions utilisés, présentons les mesures de performances utilisées pour évaluer les résultats.

3.2 Les mesures de performances

Pour la détection d'objet, il est important de garder en tête qu'il y a trois mesures de performances à garder en tête. En effet, en détection d'objet il faut être capable de bien classifier l'objet détecté mais aussi être capable de bien le localiser et dans le contexte de notre projet, on calcule aussi le temps d'inférence moyen. Nous allons donc présenter les deux mesures de performances suivantes :

- Le AP *(Average Precision*)
- L'IoU (*Intesection over Union*)

Nous ne présentons pas le calcul du temps d'inférence, puisqu'il se résume à utiliser l'horloge interne de l'ordinateur pour connaitre le temps d'exécution du code de l'inférence pour chaque image.

A noter qu'on a choisi comme mesure de classification l'AP, plutôt que le mAP (*mean Average Precision*) car nous n'utilisons actuellement qu'une seule classe. L'utilisation du mAP n'apporte donc aucune information.

Le AP est la mesure utilisée pour mesurer les performances en classification du détecteur. Pour cela, on calcule la précision et le recall. La précision, dans ce contexte, calcule le taux de faux positif, ainsi, plus le score en précision est proche de 1, plus il est probable que l'objet détecté positivement soit réellement un oiseau.

$$Precision\ rate = \frac{Nombre\ d'oiseaux\ détectés}{Nombre\ d'objets\ détectés} \tag{3.1}$$

Dans cette formule, on différentie oiseau et objet de la manière suivante, un oiseau est un vrai positif tandis qu'un objet est une prédiction positive réalisée par notre modèle.

Le recall calcule le taux de faux négatif, donc plus le score en recall est proche de 1, plus il est probable que tous les oiseaux soient correctement identifiés par le modèle.

$$Recall\ rate = \frac{Nombre\ d'oiseaux\ détectés}{Nombre\ total\ d'oiseaux\ dans\ l'image} \tag{3.2}$$

Ces deux taux permettent ainsi de quantifier la précision en classification de nos modèles. Cependant, ces deux taux dépendent aussi d'un paramètre que nous nommons score_threshold. En effet, pour chaque objet détecté, notre modèle lui attribue un score de détection. Cette détection est ainsi considérée comme une prédiction positive que si le score de détection est supérieur au seuil score_threshold. Dans la suite du rapport, nous travaillons avec un score_threshold de 0.5.

Pour la mesure en performance de localisation, nous avons choisi d'utiliser l'IoU. Cette mesure est très utilisée dans la littérature et reste très simple, on compare la surface de la boite prédite à la surface de la boite ground-truth :

$$IoU = \frac{Surface\ d'intersection}{Surface\ de\ l'union} \tag{3.3}$$

On définit la surface d'intersection comme étant la surface en commun entre la boite prédite et la boite ground-truth. De la même manière, on définit la surface d'union comme étant la surface des deux boites. L'IoU permet donc de quantifier à quelle point ces deux boites se chevauchent.

De la même manière qu'avec les mesures de performances en classification, on définit un seuil de performance, que l'on nomme iou_threshold. Ce seuil définit à partir de quelle valeur d'IoU est ce qu'on considère une détection comme étant un vrai positif ou non. Nous avons décidé d'utiliser un seuil de 0.5 comme dans la littérature. Maintenant que nous avons bien établi nos mesures de performances, nous allons présenter les performances de nos modèles, en commençant par les modèles SSD Mobilenet.

3.3 Résultats et analyse de l'architecture SSD Mobilenet.

Comme nous l'avons indiqué plus haut, nous avons commencé à travailler avec l'architecture SSD Mobilenet v2 COCO. Pour rappel, à ce moment du projet, Lockbird voulait mettre l'accent sur le fonctionnement en « temps réel » en utilisant un système embarqué. Nous avons donc décidé de privilégier le modèle le plus rapide et le plus compacte en essayant de minimiser les pertes en performances.

Le premier modèle que nous allons présenter a été entrainé à partir des images de la base PreparedData et nous avons eu les résultats suivants :

Tableau 3-1 : Performances du modèle SSD1

Métrique	**Scores**
AR rate (*Average Recall rate*)	24%
AP rate (*Average Precision rate*)	25%
AIoU score (*Average Intersection over Union score*)	39%
AIT (*Average Inference Time*)	0.078 secondes

Ce modèle a été entrainé surtout pour me permettre de me familiariser avec l'API Tensorflow. Le temps d'inférence de ce modèle sous entends une utilisation en temps réel envisageable, mais il reste des problèmes au niveau des performances en précision.

Figure 3.4 : Exemple de résultats d'inférence de notre premier modèle SSD

Sur cet exemple de la Figure 3.4, les carrés noirs représentent les boites ground-truth. Les boites rouges représentent les prédictions réalisées avec notre modèle et affichent donc le score de détection. Enfin les boites vertes représentent les détections de notre modèle qui

possèdent un score d'IoU supérieur à l'IoU_threshold (égale à 0.5). Les boites vertes affichent ainsi le score d'IoU obtenu pour cette boite.

Ainsi, on constate plusieurs problèmes sur cet exemple. Tout d'abord, même si le modèle semble correctement identifié une bonne partie des oiseaux volants, la plupart des boites ne sont pas suffisamment bien positionner pour être considérer comme un vrai positif. De plus, la taille des boites semble non adaptée aux oiseaux à détecter.

Deuxième problème qu'on peut constater, le modèle semble avoir des difficultés à repérer les oiseaux au sol. Ceci n'est pas étonnant, puisque la base d'entrainement comporte très peu d'oiseaux au sol. Dans les autres bases de données, nous avons pu corriger ce problème.

Enfin, le problème le plus contraignant reste les faux positifs de classification.

Figure 3.5 : Exemple de faux positif (droite) et d'images entrainements (gauche) pour SSD1

Comme on peut le voir sur la Figure 3.5, on constate des faux positifs dès qu'il y a des changements de contrastes, généralement un point blanc dans un fond vert ou marron. On suppose que l'origine vient des images d'entrainements comme celle de droite. On constate que sur beaucoup d'images de l'ensemble d'entrainement, les mouettes sont souvent juste des points blancs dans un fond vert, ou marron pour les peu d'oiseaux au sol.

Pour corriger ces problèmes, nous avons décidé d'ajouter plus d'images adaptés à notre utilisation. Ces images proviennent du site Waterloo et permet l'ajout d'oiseaux au sol dans la base d'entrainement. Nous n'avons cependant pas encore commencé à travailler sur les hyperparamètres. À ce moment du projet, nous avons voulu chercher à quantifier l'importance de la qualité des annotations, dans le but de mieux guider nos annotateurs et avoir une meilleure gestion qualité de leur travail.

Ainsi, pour le modèle SSD2, nous avons utilisé la base Waterloo Winter comme base d'entrainement obtenu grâce aux images d'un de nos sites, ainsi que les annotations fournis par les annotateurs de l'entreprise. Nous avons ainsi eu les résultats suivants :

Tableau 3-2 : Performances du modèle SSD2

Métriques	**Scores**
AR rate	14%
AP rate	19%
AIoU score	54%
AIT	0.13 secondes

Il est intéressant de noter que les performances en précision ont chutées mais les performances en localisation semblent s'améliorer. On explique cette différence par la difficulté des images utilisées dans l'entrainement.

Figure 3.6 : Exemple d'inférence pour le modèle SSD2

Comme on peut le voir sur la Figure 3.6, le nombre d'oiseaux présents sur l'image a beaucoup augmenté comparer à la base PreparedData. De plus, la taille des oiseaux présents sur l'image rend la détection d'autant plus complexe. En effet, la taille moyenne des oiseaux sur l'image est d'environ 30 pixels, ce qui rend la détection avec SSD beaucoup plus difficile. Ainsi, même si le modèle a l'air de se comporter de la même manière que sur la base d'images précédente, le nombre d'oiseaux sur chaque image affecte beaucoup plus le calcul des taux. On constate tout de même que d'un point de vue localisation, le modèle se débrouille convenablement même s'il est loin d'être parfait.

Cependant, ce n'est pas pour autant que nous avons réussi à corriger les problèmes de faux positifs.

Figure 3.7 Exemple de faux positifs dans SSD2

Comme on peut le constater sur la Figure 3.7, il reste des faux positifs que nous supposons lier au même problème que mentionné précédemment. En effet, même si la présence de neige semble faciliter la détection des oiseaux noirs, la détection des mouettes est encore plus complexe maintenant. Le plus contraignant avec ce problème de faux positif reste le score de détection. En effet, on constate que le faux positif (au milieu de l'image) a un score de détection de 99%. Avec cette information, on en conclut que la qualité des annotations a son importance, surtout en milieu complexe, mais que c'est loin d'être suffisant pour atteindre les performances attendues.

Pour augmenter les performances en localisation, nous avons donc décidé de travailler sur les hyparamètres notamment sur l'optimisation des ancres et modifier la taille des images d'entrées. Travailler sur l'optimisation des ancres est évidente, on constate plusieurs fois dans les différents tests d'inférence que la taille des boites n'est souvent pas approprié et cela

pourrait expliquer le mauvais positionnement de ces dernières ou au moins pourquoi certains petits oiseaux ne sont pas détectés. Cependant, nous n'avons pas réussi à parfaitement optimisées la taille des ancres. Dans le fichier config du modèle SSD, nous n'avions comme choix que le nombre d'ancres voulus et la taille minimum et maximum. Ce dernier point est contraignant car nous cherchions à limiter le nombre de couche, pour éviter de trop allonger le temps d'inférence et d'entrainement, tout en voulant choisir la taille des ancres de chaque couche.

Travailler sur la taille des images pouvaient avoir des impacts importants sur les performances du modèle. En effet, chaque image est redimensionnée avant d'être utilisé par l'architecture SSD. Or la taille des images d'entrées est de 300×300 par défaut. Il y avait donc potentiellement une grosse perte de données dans la compression des images (pour rappel, nos images sont de tailles 3840×2160). Nous avons donc cherché à trouver une taille d'entrée d'image appropriée mais nous avons été très rapidement limité par notre puissance de calcul.

Pour ce qui est des faux positifs, nous avons décidé d'utiliser le module de *hard negative mining* disponible dans l'architecture SSD. Généralement, pour pouvoir entrainer un modèle correctement, il faut fournir des images sans objet. Cela permet au modèle de s'entrainer à l'identification du *background*. L'avantage de l'architecture SSD est que chaque morceau de l'image n'ayant pas été annoté peut être considéré comme un exemple d'entrainement pour l'arrière-plan. Cependant, pour permettre un entrainement plus efficace, on ne prend pas n'importe quelle partie de l'image mais un exemple difficile, qui pourrait induire le modèle en erreur. L'architecture SSD propose de choisir combien d'exemple difficile d'objets « négatifs » (*background*) par exemple d'objets « positifs » (ici un oiseau). Nous avons modifié ce paramètre pour proposer plus d'exemples négatifs dans l'espoir de diminuer les faux positifs avec un ratio de 5 négatifs pour 1 positif, le ratio standard étant de 3 pour 1.

On note aussi que pour comprendre l'impact de chaque hyperparamètre, nous avons réalisé l'entrainement de plusieurs modèles que nous ne présenterons pas. Ainsi, même si le choix

des hyperparamètres à modifier est basé sur une recherche dans la littérature, leurs valeurs sont choisies empiriquement. Comme l'entrainement de ces modèles a pris un certain temps nous avons pu établir la base Winter+Summer. L'objectif de cette base de données était d'entrainer un modèle robuste à la météo (en tout cas à la neige et le soleil). Ainsi, pour tous les modèles entrainés sur la base Winter+Summer, nous présenterons des résultats d'inférences pour des images d'hiver et d'été, les performances étant souvent très différentes (voir Figure 3.8 pour un exemple d'inférence réalisée).

Ainsi, en choisissant un nombre de 6 couches compris entre [0.05 ;0.5] et une taille d'images d'entrées de 800*800, nous avons eu les résultats suivants :

Tableau 3-3 : Performances du modèle SSD3

Métriques	**Scores**
AR rate	39%
AP rate	34%
AIoU	50%
AIT	0.447 secondes

Les résultats obtenus sont pour le moins décevant. L'augmentation en performance de précision est intéressante mais loin d'être satisfaisante, surtout qu'il y a une baisse de performance en localisation. Le temps d'inférence a légèrement augmenté, sans surprise, mais reste largement satisfaisante selon nos attentes.

Figure 3.8 : Exemples d'inférences pour le modèles SSD3 sur la base Winter+Summer

En regardant les résultats d'inférence, on constate que le choix des tailles d'ancres a bien permis la détection des oiseaux plus petits. Cependant, cela n'a pas résolu le problème de localisation des ancres. De plus, on peut voir sur certain cas que même si la détection est bien classifiée et localisée, la forme de la boite n'est pas appropriée. Enfin, le problème de faux positif continue de persister, même si nous avons réussi à diminuer la quantité. Compte tenu des résultats obtenus, l'architecture ne semble vraiment pas adapté à nos utilisations. En effet, la taille des images d'entrées de l'architecture est forcément de taille carrée. Comme nos images ont un aspect ratio de 16:9, il y a donc tout le temps de la distorsion d'images. Ceci ne serait pas un problème si nous possédions des machines suffisamment puissantes pour atteindre une taille significative pour nos images d'entrée. Or, avec nos machines nous sommes limités à une taille d'entrée de 800*800. L'architecture étudié ne semble donc pas pouvoir remplir nos attentes. Après en avoir discuté avec l'entreprise, nous avons décidé de moins mettre l'accent sur le fonctionnement en temps réel et plus se concentrer sur les performances du modèle. Pour cela, nous avons donc décidé d'étudier les performances de l'architecture Faster R-CNN Inception V2.

3.4 Résultats et analyse de l'architecture Faster R-CNN

Pour le premier modèle à tester avec l'architecture FRCNN, nous avons décidé de tester l'hypothèse de (Jonathan Huang et al., 2017) sur le lien entre le nombre de propositions de régions et le temps d'inférence. Nous avons donc limité le nombre de propositions de régions du modèle à 50 et nous avons obtenu les résultats suivants :

Tableau 3-4 : Performances du modèle FRCNN1

Métriques	**Scores**
AR rate	41%
AP rate	46%
AIoU	53%
AIT	1.066 secondes

Sans grande surprise, le modèle FRCNN1 a de meilleures performances en précision que nos modèles sur l'architecture SSD. Le plus intéressant reste que le temps d'inférence moyen est proche de 1 seconde. L'architecture est donc envisageable pour nos applications.

Figure 3.9 : Exemples de résultats d'inférence pour FRCNN1

En ce qui concerne les résultats, on constate qu'il y a moins de faux positifs que pour l'architecture SSD et surtout qu'ils ont aussi un score de détection moins élevé. En contrepartie, il y a moins de détection. Cependant, on constate dans l'image d'été de la Figure 3.9¸ qu'une des boites détectées ne correspond pas à la boite ground-truth car elle est trop grande. Ainsi, nous supposons que cette différence en nombre de détection est en partie dû à l'utilisation de boites non optimisées pour notre application. La plupart des ancres doivent donc être trop grande et ne permettent donc pas la détection des oiseaux. Pour nous assurer de pouvoir détecter suffisamment d'oiseaux, nous décidons de passer le nombre maximal de proposition à 125. L'objectif est de pouvoir au moins détecter 100 oiseaux sur nos sites. Nous supposons que l'augmentation en temps lié à cette augmentation est acceptable.

Nous avons donc commencé sur notre second modèle pour l'architecture Faster R-CNN. Pour ce second modèle, nous avons décidé de commencer par optimiser le réseau de détection. Pour cela, nous avons suivi la méthodologie de SSD en commençant par optimisé la taille des ancres de détection ainsi qu'en modifiant la taille des images d'entrées. En ce qui

concerne cette dernière, l'architecture FRCNN présente l'avantage de pouvoir conserver l'*aspect ratio* de nos images. Nous sommes toujours limités par la taille des images, mais au moins il n'y a pas de distorsion dans l'image.

Nous avons décidé de modifier la taille de l'ancre de base ainsi que les échelles pour que la plupart des détections réalisées se concentrent sur des objets de petites tailles. Après avoir annoté plusieurs centaines d'images de Waterloo, nous nous sommes rendu compte que la tailles des oiseaux étaient relativement constante. En effet, la distance de la caméra fait que les changements de tailles entre les oiseaux de la même espèce sont négligeables. Cette ressemblance de taille se retrouve aussi entre les espèces d'oiseaux quand ils ne sont pas sur le même plan. Nous avons donc choisi comme échelle les tailles d'oiseaux les plus répandues.

Nous avons aussi essayé de régler le problème lié au regroupement d'objet. En effet, nous avons indiqué dans la revue de littérature que les algorithmes de détection d'objets ont du mal avec des objets regroupés en *cluster*. Nous avons donc décidé de réduire le *stride* du détecteur.

On obtient ainsi les résultats suivants :

Tableau 3-5 : Performances du modèle FRCNN2

Métriques	**Scores**
AR rate	64%
AP rate	51%
AIoU	62%
AIT	2.622 secondes

Figure 3.10: Exemples de résultats pour FRCNN2

Ainsi, optimiser la taille des ancres semblent avoir porté ses fruits. On a réussi à booster toutes les performances tout en respectant nos contraintes de temps.

On constate sur la Figure 3.10 que le nombre d'oiseaux détecter a effectivement augmenté, mais surtout que le modèle est maintenant capable de détecter de plus petits oiseaux. Le modèle n'est pas encore capable de détecter tous les oiseaux des images de Winter. Ce n'est pas vraiment un problème majeur pour l'entreprise. En effet, dans le contexte d'utilisation de l'entreprise, être capable de repérer un oiseau d'un groupe peut être suffisant pour effrayer tout le groupe. Ainsi, repérer tous les oiseaux n'est pas encore la plus grande priorité, on veut juste être capable de repérer une majorité des oiseaux. On constate cependant de nouveau des faux positifs, notamment quand il y a des objets blancs sur les images d'été. On suppose que ces faux positifs proviennent des images d'hiver. La présence de neige et de terre aurait pu induire notre modèle en erreur, surtout avec les mouettes. Cependant, l'architecture FRCNN ne propose pas de module pour le *hard negative mining*. Pour résoudre les problèmes de faux positifs, nous avons tenté d'améliorer les performances en modifiant les poids du *weight decay* ainsi qu'en implémentant une régularisation L2. En effet, même si (S. Ren, 2015) ont rendu possible la mise en place d'une régularisation L2, ils ne l'ont pas implémenté dans leur modèle. Nous avons donc réalisé plusieurs entrainements pour déterminer un coefficient adapté à notre utilisation.

Pour FRCNN3, nous avons obtenus les résultats suivants :

Tableau 3-6 : Performances du modèle FRCNN3

Métriques	**Scores**
AR rate	91%
AP rate	95%
AIoU	86%
AIT	3.023 secondes

Ce modèle satisfait enfin les attentes de l'entreprise. Nous avons réussi à atteindre des performances en précision et localisation qui permettrait une application en terrain. La Figure 3.11 est un exemple de résultats obtenus.

Figure 3.11: Exemples d'inférence pour FRCNN3

Le modèle n'est toujours pas en mesure de détecter tous les oiseaux des images d'hiver, mais ceci n'est pas très surprenant. Les oiseaux sur les images d'hiver sont regroupés et plus petits ce qui est particulièrement difficile pour les algorithmes de détection d'objet. Avec les limites de calculs que nous avons pour ce projet, nous considérons donc que les performances sont satisfaisantes. Cependant, il reste tout de même certains points à améliorer.

Le premier point important à garder en tête est qu'actuellement, notre modèle n'est entrainé que pour le site de Waterloo. Ceci est inévitable car nous n'avons pas d'images annotées des autres sites. Ainsi, comme il reste quelques faux positifs dans les inférences, notamment avec les images d'hiver, il est envisageable que dans un site avec plus de déchet sur le sol, le modèle n'arrive pas à distinguer correctement les déchets des oiseaux. De plus, nous avons optimisées les tailles des ancres pour refléter les tailles d'oiseaux les plus répandues. Ces tailles sont entièrement dépendantes de la distance des oiseaux à la tourelle et ne sont donc surement pas transposable à un autre site.

Le second point d'amélioration possible serait d'arrivé à une détection de tous les oiseaux présents sur l'images, surtout pour les images d'hivers. En effet, même si les mesures en performances sont satisfaisantes pour le projet, elles ne permettent pas encore une application parfaite sur le terrain puisqu'on constate encore des oiseaux non détectés par le modèle. Pour arriver à ce résultat, il serait plus évident d'avoir plus de puissance de calcul ce qui nous permettrait de moins limités le nombre de proposition de région. Il faudrait cependant aussi modifier les différents poids dans le calcul de la fonction de *loss*, mais nous avons trouvé peu de documentation sur ces variables et n'avons malheureusement pas eu le temps de tester différentes valeurs.

On peut réaliser ce même constat pour la localisation des boites. De la même manière, il aurait été intéressant d'optimiser les poids du régresseur mais n'ayant trouvé ni de

documentation dessus ni eu le temps de tester empiriquement les valeurs, nous avons décidé de laisser les valeurs de base.

3.5 Conclusion

En conclusion, nous pouvons commencer par rappeler l'ensemble des résultats que nous avons trouvé au cours de ce projet de session.

Tableau 3-7 : Récapitulatif des résultats obtenus

Modèle	AR rate	AP rate	AIoU	AIT
SSD1	24%	25%	39%	0.078 secondes
SSD2	14%	19%	54%	0.13 secondes
SSD3	39%	34%	50%	0.447 secondes
FRCNN1	41%	46%	53%	1.006 secondes
FRCNN2	64%	51%	62%	2.622 secondes
FRCNN3	91%	95%	86%	3.023 secondes

En optimisant la taille des ancres de détection, et en agrandissant la taille des images d'entrées, nous avons pu optimiser les performances de notre modèle pour arriver aux performances désirées par l'entreprise. Ces performances sont loin d'être parfaites mais remplissent les critères établis par l'entreprise.

Cependant, ces optimisations ne permettent pas l'utilisation de nos modèles sur d'autres sites. Nous ne considérons pas que cela comme un problème majeur, puisque nous n'avons pas de données sur les autres sites. Une fois une base de données suffisamment conséquente, l'entrainement d'un modèle généraliste sera envisageable.

CONCLUSION

Ce document vient présenter le travail réalisé pour l'entrainement d'un modèle de détection d'oiseau pour l'entreprise Lockbird. Ce modèle devait être en mesure de détecter une majorité des oiseaux présents sur une image pour permettre à notre système d'éloigner les oiseaux de manière autonome.

Après la revue de littérature, nous avons pu réduire le choix des architectures à deux, l'architecture Faster R-CNN et l'architecture SSD. Les contraintes du projet font qu'il n'y avait pas de modèle plus adapté pour notre application que l'autre. Après plusieurs entrainements, nous avons conclu que pour notre application, l'architecture Faster R-CNN était plus adapté. En effet, les images utilisées sont complexes et présentent beaucoup d'exemples négatifs difficile (*hard negative examples*) nécessitant un modèle robuste en classification.

L'obstacle du projet a donc été d'optimiser notre modèle suffisamment pour que l'inférence se passe en moins de 5 secondes sans trop perdre en précision. Pour cela, nous avons donc chercher à optimiser la génération des ancres ainsi que la convergence de la fonction d'objectif (*loss function*). Nous avons ainsi réussi à obtenir un modèle satisfaisant les attentes de l'entreprise, mais il faut rester vigilant. En effet, même si le modèle a de bon résultats sur l'inférence, il reste optimiser et entrainer que pour un site. Pour rendre ce modèle exportable à d'autres sites, il nous faut encore obtenir des images ainsi que réaliser l'entrainement avec ces images.

De plus, il serait aussi intéressant pour l'avenir de l'entreprise de commencer à étudier l'entrainement d'un modèle multi-classe, ou chaque oiseau est identifié par son espèce. Cette méthodologie permettrait d'avoir plus d'information concernant l'origine des faux positifs tout en permettant un choix spécifique par espèce de système d'effarouchement, comme des cris d'oiseaux d'une certaine espèce. L'inconvénient pour l'entreprise est qu'il faudra réannoter toutes les images des différentes bases d'entrainement ainsi qu'un changement de

méthodologie dans la création de base de données pour permettre une répartition équitable des annotations en fonction de l'espèce. Pour ces raisons, l'entreprise n'a pas pu permettre le développement de cette approche.

ANNEXE I

MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL

Pour la mise en place de Tensorflow Object Detection API, nous avons suivi les instructions de (Mirza, 2018) que nous allons détaillé ici.

Pour mettre en place Tensorflow API, il faut commencer par télécharger Visual Studio 2017. Nous recommandons la version 2017 plutôt que la version 2019 car cette dernière ne possède pas encore les fonctionnalités pour être utiliser avec les autres logiciels que nous allons installer. Une fois l'installation réalisée, il est important de s'assurer que les composants pour C++ développement et Windows 10 SDK (Version 10.0.15063.0) sont aussi installées.

Il faut ensuite installer NVIDIA GeForce Experience (https://www.geforce.com/geforce-experience/download). Ce logiciel a pour objectif de s'assurer que les drivers NVIDIA sont bien à jour.

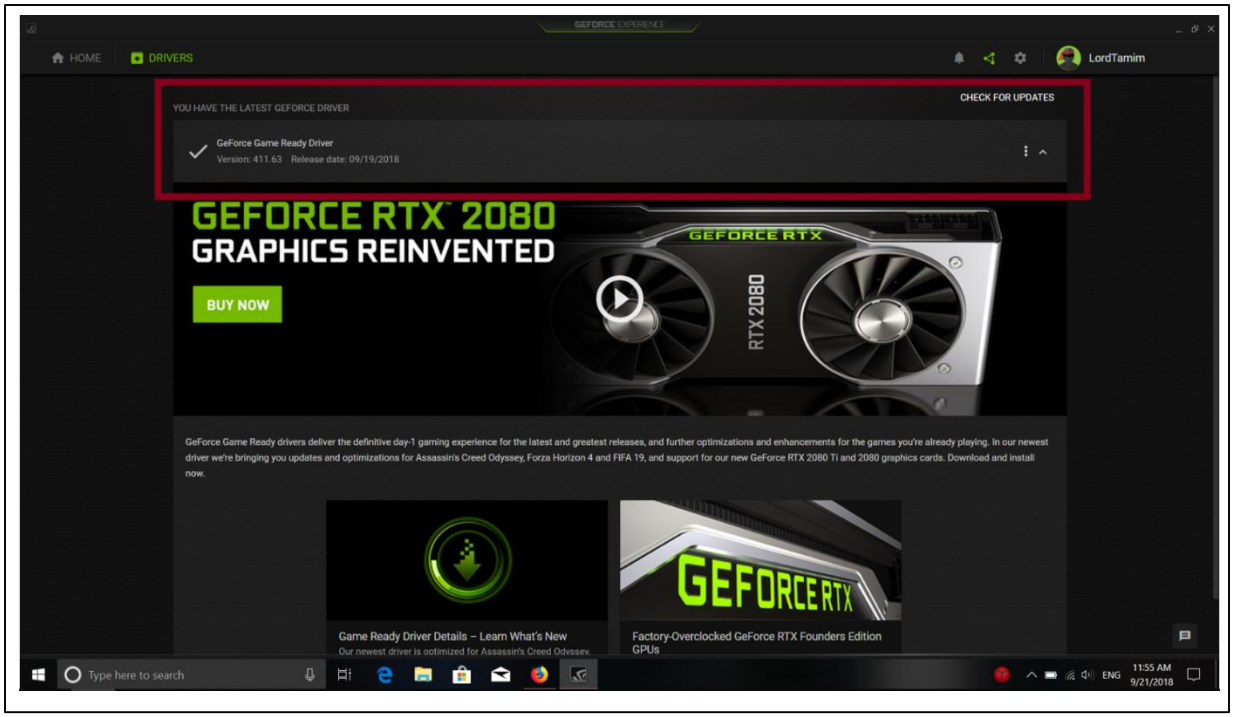


Figure A.I.1: Indicatif de driver à jour

Cette étape peut nécessiter la création d'un compte NVIDIA. Même si ce logiciel n'est installé que pour s'assurer que les drivers sont à jour, il est important de faire cette vérification pour le fonctionnement des autres logiciels.

Une fois les mises à jour faite, il faudra installer NVIDIA Cuda Toolkit 10.0 (https://developer.nvidia.com/cuda-10.0-download-archive?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exelocal). Ce driver est primordial car il permet à Tensorflow API d'utiliser la carte GPU (*Graphic Processing Unit*).

Pour vous assurer que l'installation s'est bien déroulé, il est possible d'effectuer un test avec Visual Studio. Pour cela, il faut aller au dossier d'installation de Cuda Toolkit, par défaut à l'adresse suivante :
C:\ProgramData\NVIDIACorporation\CUDASamples\v10.0\1_Utilities\deviceQuery.
Une fois à cette adresse, il faut choisir l'exécutable VS Solution associé à la version de Visual Studio installée (soit la version 2017).

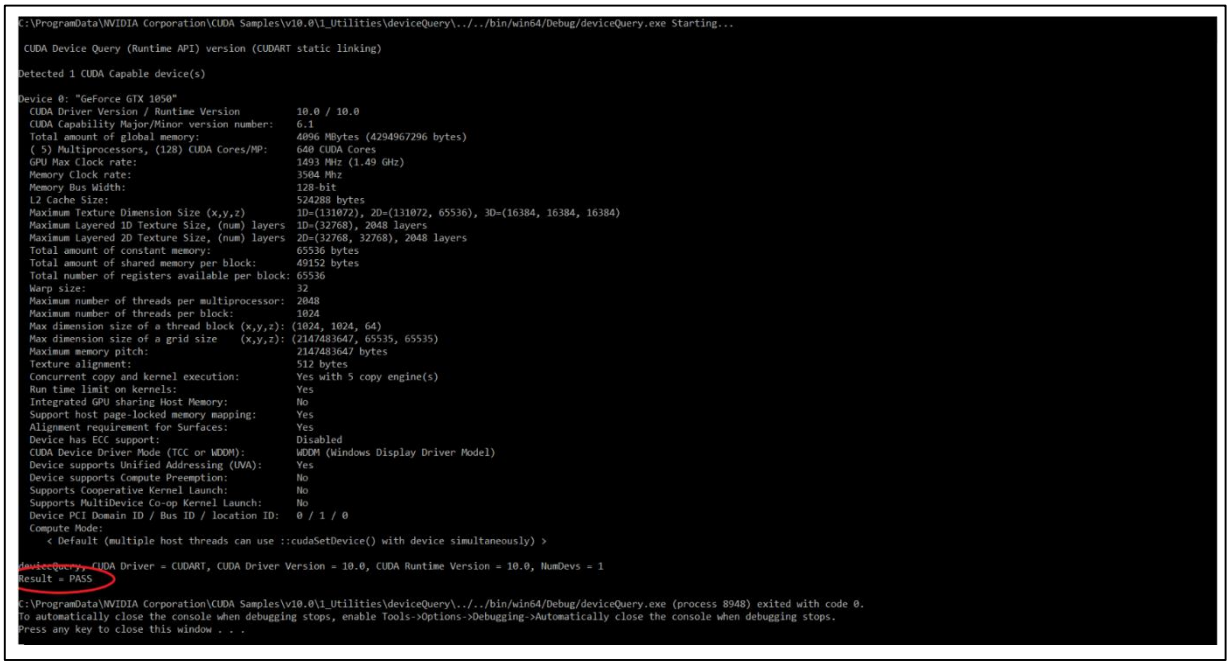


Figure A.I.2: Vérification de l'installation de Cuda Toolkit

Si la vérification est positive, il faut maintenant permettre l'accès de ce logiciel de n'importe quel dossier. Pour cela, il faut entrer les 3 adresses suivantes comme variable d'environnement de Path :

- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\include

Il est important de s'assurer que vous avez bien défini ces trois adresses comme variable de système et non juste pour l'utilisateur. Cela a pour but d'éviter des problèmes de permissions.

Une fois CUDA Toolkit disponible comme variable d'environnement, il faut installer la librairie cuDNN v7.X pour CUDA Toolkit v.10 (https://developer.nvidia.com/cudnn). Cette librairie contient toutes les opérations nécessaires aux réseaux de neurones. Une fois l'installation finie, il faut ajouter la librairie au fichier de CUDA Toolkit. Pour cela, il faut donc :

- Copier le fichier \cuda\bin\cudnn64_7.dll à l'adresse C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin
- Copier le fichier \cuda\include\cudnn.h à l'adresse C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\include
- Copier le fichier \cuda\lib\x64\cudnn.lib à l'adresse C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64.

Maintenant que l'environnement de travail a été mis en place pour permettre l'utilisation de la carte GPU, il ne reste plus qu'à installer Python et Tensorflow. Commencons donc par installer Python à l'aide d'un interpreter, plus précisément Anaconda 64-bit Distribution (https://www.anaconda.com/distribution/). Ce logiciel permet la gestion des packages de python et est complètement open-source. Pour nous assurer que le logiciel est vraiment à jour, il faut exécuter les commandes suivantes dans un terminal :

- conda update conda
- conda update anaconda

La raison pour laquelle nous avons choisi Anaconda est que le logiciel permet la création d'environnement spécifique. Il est donc possible d'installer différentes versions de Python avec différents packages en créant juste un nouvel environnement.

Pour cela, nous allons maintenant détailler la procédure de création d'un environnement ainsi que l'installation de Python 3.7. Après avoir lancé Anaconda, dans l'onglet Environnement, cliquer sur « Create » pour pouvoir créer et paramétrer un nouvel environnement.

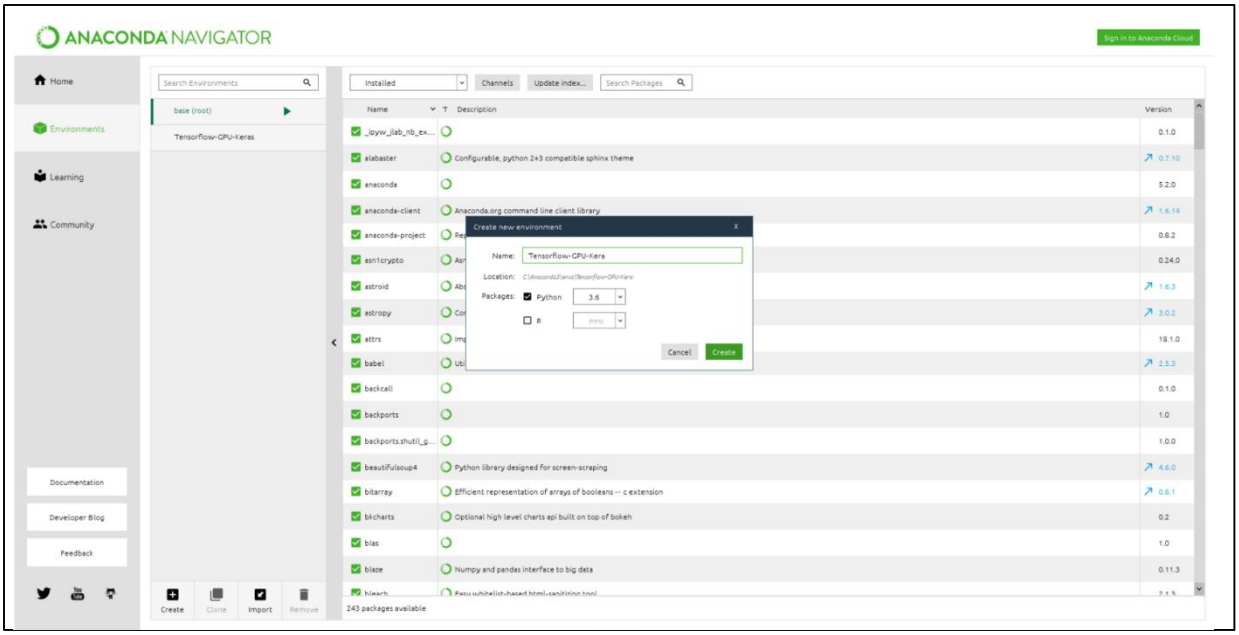


Figure A.I.3 : Création d'un environnement Anaconda

Le choix du nom reste libre, mais il est important de choisir la bonne version de Python. Pour l'application souhaité, il est important de choisir Python 3.7. Une fois l'environnement crée, cliquer sur la flèche verte de l'environnement et sélectionner « Open Terminal ». C'est à travers ce terminal que nous pouvons exécuter les commandes sur l'environnement. Ainsi, dans le terminal, il ne reste plus qu'à exécuter la commande suivante

- pip install tensorflow-gpu

Pour vous assurer que l'installation a été réussite, dans ce même terminal taper successivement les commandes suivantes :

- python
- import tensorflow
- import keras

Si les deux commandes s'exécutent sans erreurs, l'installation a été réussite.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. & Torralba, A. (2015). Object detectors emerge in Deep Scene CNNs. International Conference on Learning Representations, May 7-9.

Fu, C. Y., Liu, W., Ranga A., Tyagi, A. & Berg. A. C. (2017). DSSD : Deconvolutional Single Shot Detector. ArXiv, 1701.06659.

Dollar, C. L. (2014). Edge Boxes: Locating Object Proposals from Edges. European Conference on Computer Vision vol. 8693, 391-405.

Girshick, R. (2015). Fast R-CNN. IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448.

Huang J. et al. (2017). Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. IEEE Conference on Computer Vision and Pattern vol. 1, 3296-3297.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904-1916.

Mirza, T. (2018, Septembre 18). Python Environment Setup for Deep Learning on Windows 10. Récupéré sur Towards Data Science: https://towardsdatascience.com/python-environment-setup-for-deep-learning-on-windows-10-c373786e36d1.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 580-587.

Ren, S., He, K., Girshick, R. & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149.

Liu W. et al. (2016) SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer.

Zhao, A., Zheng, P., Xu, S. & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems. Early Access. doi: 10.1109/TNNLS.2018.2876865.