Rapport Bilbiographique

Etude de la Localisation Active d'Objets par Apprentissage par Renforcement Profond

- Encadré par :

  Pr. BLOCH Isabelle

- Réalisé par :

  RAMOUL Samy Rayan : *raysamram@gmail.com*

23/11/2020

Chapitre 1

Etat de l'art

1.1 Introduction à la détection d'objets

La reconnaissance d'objets est un terme général pour décrire un ensemble de techniques de vision par ordinateur combinées aboutissant à l'identification d'objets dans des photographies numériques.

Nous pouvons distinguer ces trois tâches de vision par ordinateur :

— Classification d'image : prédire le type ou classe d'un objet dans une image ne contenant qu'un unique objet.

— Localisation d'objets : déterminer l'emplacement d'un ou plusieurs objets dans une image et les encadrer.

— Détection d'objets : localiser la présence d'objets avec un cadre de sélection et les types ou classes des objets localisés dans une image.
En entrée on a donc une image contenant un ou plusieurs objets, comme une photographie. En sortie une ou plusieurs boîtes englobantes (ou "bounding box") et une étiquette de classe pour chaque boîte englobante.

La classification d'image implique la prédiction de la classe d'un objet dans une image. La localisation d'objet fait référence à l'identification de l'emplacement d'un ou plusieurs objets dans une image et à la définition d'un rectangle englobant autour de l'objet visé. La détection d'objets combine ces deux tâches et localise et classe un ou plusieurs objets dans une image.
Une autre extension de ce domaine des tâches de vision par ordinateur est la segmentation d'objet, également appelée «segmentation d'instance d'objet» ou «segmentation sémantique», où les instances d'objets reconnus sont indiquées en mettant en évidence les pixels spécifiques de l'objet au lieu d'un cadre de délimitation grossier.
À partir de cet éventail de techniques, nous pouvons voir que la reconnaissance d'objets fait référence à une suite de tâches de vision par ordinateur difficiles.
La détection d'objets est un des problèmes fondamentaux et les plus présents dans le domaine de la vision par ordinateur ; l'objectif étant de pouvoir à partir d'une image déterminer quelles classes d'objets y sont présentes et en quel nombre (voitures, humains ..etc.) et afin de réaliser cet objectif plusieurs méthodes ont été proposées et dont les plus importantes seront exposées dans ce premier chapitre.

1.2 Méthodologies appliquées pour la détection d'objets

Des méthodes appliquées à ces problématiques de vision vont être présentés dans cette partie, certaines donnant de très bon résultats (Approches par réseaux de neurones convolutionnels seuls ou combinés à un système d'attention ou de l'apprentissage par renforcement) d'autres proposant des approches innovantes de représentation du problème mêlés à des classifieurs (Modèles partiels, fenêtre glissante, histogramme orienté des gradients).

1.2.1 Évaluation de la détection

Afin de déterminer l'efficacité des modèles de détection d'objets on se base sur la méthode dites d'accuracy (équation 1.1) et ce après un test sur un jeu de données comme par exemple Pascal Visual Object Classes Challenge [16] ou ImageNet.

Ensuite, il est important d'introduire une mesure n'étant pas sensible à la distribution non-uniforme des classes de ces jeux de données donc utiliser une mesure par accuracy simple ne serait pas efficace, de plus il est important que cette mesure permette d'évaluer le degrés de confiance en la prédiction pour chaque *bounding box* définie. La précision comme définie dans l'équation 1.1 mesure le taux de faux positifs ou ratio de bons objets détectés par rapport au nombre total d'objets que le classifieur à détecter.

$$précision = \frac{VraiPositif}{VraiPositif + FauxPositif} \tag{1.1}$$

Le rappel quant à lui (équation 1.2) ou *"recall"* en Anglais, fait lui référence au taux de faux négatifs ou le ratio d'objets détectés correctement par rapport au nombre total d'objets dans le jeu de données.

$$rappel = \frac{VraiPositif}{VraiPositif + FauxNégatif} \tag{1.2}$$

Ces deux équations sont liés à un seuil connecté au modèle qui définit à partir de quelle valeur (ou taux de confidence) le classifieur déterminera qu'un objet appartient à une classe. Ces deux concepts sont utilisés afin de calculer une mesure qui a été introduite [16] afin de répondre aux problèmes de biais cités précédemment résultant en la mesure d'*average precision* ou *AP* (équation 1.3), où $Rappel_i$ pouvant prendre 11 valeurs $[0, 0.1, 0.2, ..., 1]$ ainsi on prend la valeur moyenne de la précision sur toutes les valeurs de rappel.

$$AP = \frac{1}{11} \sum_{Rappel_i} Précision(Rappel_i) \tag{1.3}$$

Le *mAP* sera quant à lui la moyenne de l'*average precision* sur l'ensemble des classes d'objets.

Cependant les mesures vues jusqu'à présent concernent le problème de classification d'objets, concernant celui de localisation d'objets il faut ajouter une définition liée aux surfaces de *bounding box* et celle de Localisation et intersection sur Union *"Localization and Intersection over Union"* définie par l'équation 1.4 a comme avantage de prendre en compte les modèles et les types de formes prédites (par exemple certains modèles peuvent localiser des objets dans une zone rectangulaire, d'autres une segmentation pixel par pixel) et cela résume à quel point l'objet à prédire empiète sur l'objet prédit par le modèle comme peut le représenter la figure 1.1.

$$IoU(b,g) = surface(b \cap g)/surface(b \cup g). \tag{1.4}$$

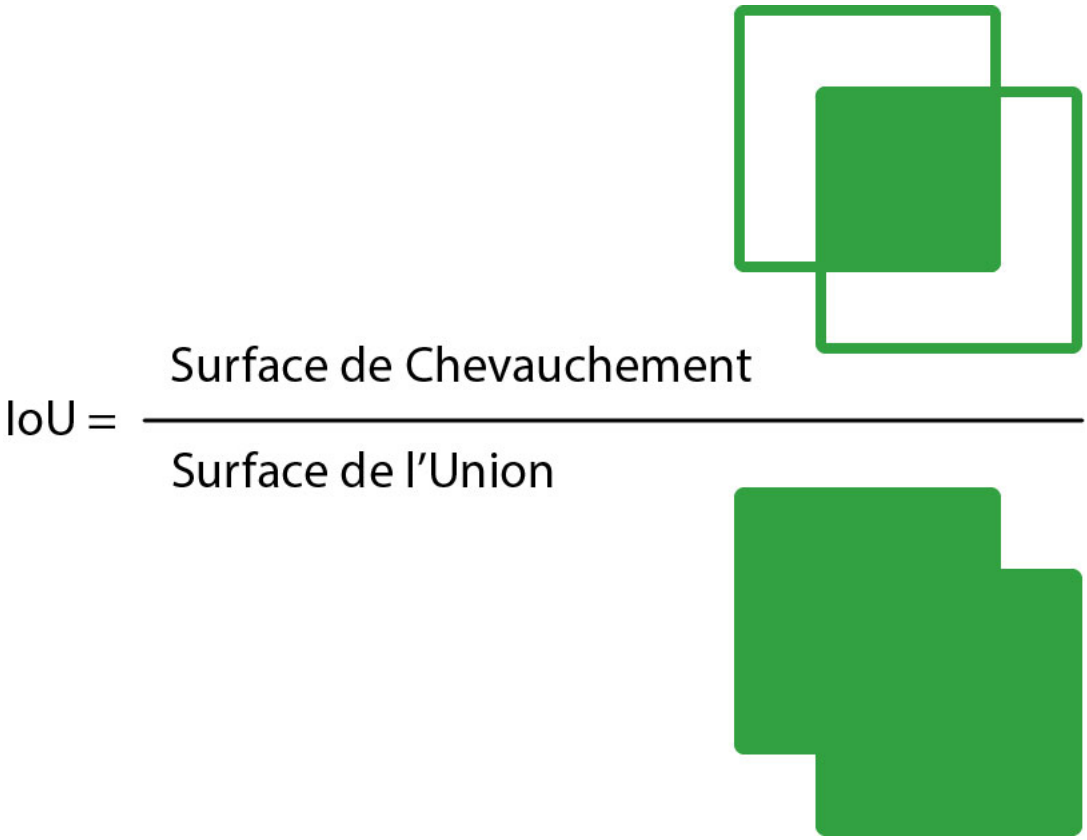


FIGURE 1.1: IoU ou Localisation et intersection sur Union

1.2.2 Méthodes appliquées

1.2.2.1 Detections par modèles partiels

Cette méthode [1] consiste à représenter chaque type d'objet comme composé d'un ensemble de sous parties appelées " Pictorial structures " qui sont des structures élémentaires à tailles différentes capturant chacune une partie différente d'un objet, créant ainsi un dictionnaire visuel sur lequel est ensuite applicable un modèle de discrimination.
Avantages :

- Permet l'analyse de similarités entre objets en déterminant des structures communes.
- Bons résultats de détection (comme le montre la figure 1.2 sur les lignes *a*, *b* et *c* qui sont plusieurs variantes du modèles testées et leurs classements dans la compétition le tout comparé avec la mesure *AP*).

Inconvénients :

- Devoir définir un ensemble de structure généralisables et efficaces pour les domaines visés.
- Plus le dictionnaire est riche, plus le modèle de discrimination doit être complexe.

		aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbik	pers	plant	sheep	sofa	train	tv
(a)	**base**	.336	.371	.066	.099	.267	.229	.319	.143	.149	.124	.119	.064	.321	.353	.407	.107	.157	.136	.228	.324
(b)	**BB**	.339	.381	.067	.099	.278	.229	.331	.146	.153	.119	.124	.066	.322	.366	.423	.108	.157	.139	.234	.328
(c)	**context**	.351	.402	.117	.114	.284	.251	.334	.188	.166	.114	.087	.078	.347	.395	.431	.117	.181	.166	.256	.347
(d)	**rank**	2	1	1	1	1	1	2	2	1	2	4	5	2	2	1	1	2	2	3	1
	(UofCTTIUCI)	.326	.420	.113	.110	.282	.232	.320	.179	.146	.111	.066	.102	.327	.386	.420	.126	.161	.136	.244	.371
	CASIA Det	.252	.146	.098	.105	.063	.232	.176	.090	.096	.100	.130	.055	.140	.241	.112	.030	.028	.030	.282	.146
	Jena	.048	.014	.003	.002	.001	.010	.013		.001	.047	.004	.019	.003	.031	.020	.003	.004	.022	.064	.137
	LEAR PC	.365	.343	.107	.114	.221	.238	.366	.166	.111	.177	.151	.090	.361	.403	.197	.115	.194	.173	.296	.340
	MPI struct	.259	.080	.101	.056	.001	.113	.106	.213	.003	.045	.101	.149	.166	.200	.025	.002	.093	.123	.236	.015
	Oxford	.333	.246					.291			.125			.325	.349						
	XRCE Det	.264	.105	.014	.045	.000	.108	.040	.076	.020	.018	.045	.105	.118	.136	.090	.015	.061	.018	.073	.068

FIGURE 1.2: Résultats de la méthode par modèles partiels sur le Pascal VOC Challenge de 2008

1.2.2.2 Méthode de fenêtre glissante

Dans les méthodes classiques un modèle binaire (ou plusieurs) permettent de déterminer si un objet est présent ou non dans une image sans déterminer une localisation, afin de palier à cette problématique il a été proposé de faire glisser une fenêtre sur l'ensemble de l'image, fenêtre qui serait évaluée en utilisant une fonction de qualité (résultante d'un classifieur par exemple), cependant cette méthode a un coût de calcul conséquent. Une solution a été proposée [2] pour réduire ce coût, l'intuition étant qu'au lieu de parcourir l'ensemble des régions de l'image les régions seraient parcourues par rapport à leurs qualités : chaque fois qu'un rectangle ou "bounding box" est définie, cette zone est divisée en sous régions puis classées selon leurs pertinences pour être parcourue à la recherche d'un objet.

Avantages :

— Technique donnant des résultats similaires aux fenêtres glissantes classiques mais en étant bien moins coûteuse.

Inconvénients :

— La méthode utilise une reconnaissance au travers de régions et objets sous formes de rectangles, son utilité est donc moindre pour les contextes et objets ne respectant pas cette forme comme pour les images biologiques par exemple.

1.2.2.3 HOG et Support Vecteur Machine

Cette méthode [3] consiste à mélanger un modèle discriminateur ainsi qu'un modèle de support vecteur machine, le tout après avoir transformé les zones d'images en représentation HOG (ou histogramme des gradients). Un modèle différent de SVM linéaire est affecté à chaque prédiction de classe puis on applique une calibration sur la sortie de l'ensemble de ces modèles afin d'en définir des probabilités d'appartenance à chaque classe

Avantages :

— Efficace même avec un petit set d'apprentissage (voir figure 1.3)

Inconvénients :

— Sensible aux variations de l'image (translation , rotation, ..etc).

Approach	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	person	pottedplant	sheep	sofa	train	tvmonitor	mAP
NN	.006	.094	.000	.005	.000	.006	.010	.092	.001	.092	.001	.004	.096	.094	.005	.018	.009	.008	.096	.144	.039
NN+Cal	.056	.293	.012	.034	.009	.207	.261	.017	.094	.111	.004	.033	.243	.188	.114	.020	.129	.003	.183	.195	.110
DFUN+Cal	.162	.364	.008	.096	.097	.316	.366	.092	.098	.107	.002	.093	.234	.223	.109	.037	.117	.016	.271	.293	.155
E-SVM+Cal	.204	.407	.093	.100	.103	.310	.401	.096	.104	.147	.023	.097	.384	.320	.192	.096	.167	.110	.291	.315	.198
E-SVM+Co-occ	.208	.480	.077	.143	.131	.397	.411	.052	.116	.186	.111	.031	.447	.394	.169	.112	.226	.170	.369	.300	.227
CZ [6]	.262	.409	–	–	–	.393	.432	–	–	–	–	–	–	.375	–	–	–	–	.334	–	–
DT [7]	.127	.253	.005	.015	.107	.205	.230	.005	.021	.128	.014	.004	.122	.103	.101	.022	.056	.050	.120	.248	.097
LDPM [9]	.287	.510	.006	.145	.265	.397	.502	.163	.165	.166	.245	.050	.452	.383	.362	.090	.174	.228	.341	.384	.266

FIGURE 1.3: Résultats de l'approche par SVM : ESVM+Cal (Exemplar-SVM with calibration) sur le challenge Pascal VOC 2007 © [3]

1.2.2.4 Réseaux de neurones Convolutionnels

Cette méthode se base sur l'utilisation d'un type de réseaux de neurones, appelé réseau de neurones convolutionnel, qui est un type de modèle orienté pour l'analyse des images et leur interprétation en utilisant une représentation sous forme de filtres et des opérations de convolutions. Ici c'est utilisé pour la localisation et détection d'objets dans une image.

Avantages :

— Méthode généralisable à un grand nombre de classes.
— Obtient des performances plus élevées que celles par des classifieurs par apprentissage statistique classique.

Inconvénients :

— Demande une puissance de calcul conséquente pour le parcours de l'ensemble de l'image par fenêtre.
— Nécessite une grande quantité de données pour être performante.

1.2.2.5 R-CNN

C'est une méthode [15] très puissante, basée sur une technique appelée " Region proposal " qui consiste à, afin d'économiser de la puissance de calcul et du temps, au lieu d'effectuer une convolution sur chaque parcelle de l'image, effectuer d'abord un pré traitement de l'image (une segmentation) afin d'en extraire les objets existants, puis fait passer l'image traitée par un réseau convolutionnel. On peut donc résumer la méthode à 3 modules :

— Définition de régions : génération et extraction de régions segmentées candidates à la classification.
— Extraction de caractéristiques : extraire les caractéristiques de chaque région et ce grâce à une architecture de réseaux de neurones convolutionnels.
— Classifieur : Étiquetage des caractéristiques en utilisant un modèle de discrimination comme par exemple un SVM linéaire.

La figure 1.4 montre ces différents modules et les interactions les liant.

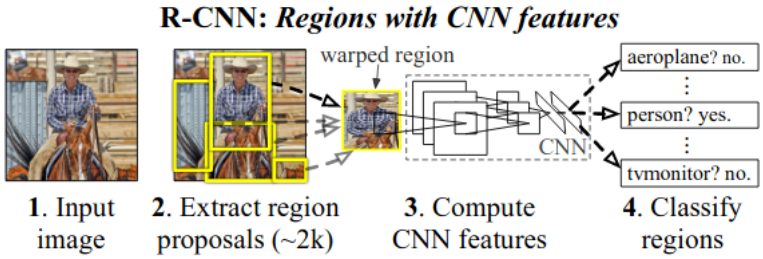


FIGURE 1.4: R-CNN © [15]

Le problème de cette méthode est sa lenteur, et plusieurs techniques ont été proposées dans la littérature afin de l'améliorer :

— *Fast R-CNN* [5] : Utilise une convolution pour la fenêtre glissante afin de classifier chaque région de l'image.
— *Faster R-CNN* [6] : Au lieu d'utiliser un algorithme de segmentation, fait appel à un *CNN* pour la création de celle-ci.

Avantages :

— Méthode donnant globalement (comme on peut observer son résultat mAP sur la figure 1.5 se classant la première sur le challenge Pascal VOC de 2010) les meilleurs résultats.

Inconvénients :

— Sensible à l'occlusion et la troncature.
— Coûteux en temps et en espace mémoire.
— La détection d'objet est lente.
— Implique la préparation et entraînement de plusieurs modules différents.

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20]†	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18]†	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

FIGURE 1.5: Résultats R-CNN sur le Pascal VOC Challenge de 2010 © [15]

1.2.2.6 Attention model

Ce modèle [7] implémente plusieurs aspects des techniques citées précédemment et est différente des autres car elle permet la génération du résumé textuel de l'image (par exemple "A bird flying over a body of water"), le modèle est composé de :

— En entrée une image.
— Un réseau de neurones convolutionnel pour l'extraction des caractéristiques de l'image
— Un recurent neural network (modèle de réseau de neurones séquentiel permettant la génération d'entrées ou de sorties de tailles variables tout en gardant une mémoire des traitements précédents) avec l'utilisation d'un modèle attention, lui permettant de se focaliser à la prédiction de chaque mot.
— En sortie des mots générées par le modèle et son attention dynamique.

Avantages :

— Permet la génération d'une sémantique de l'image et d'un contexte.

Inconvénients :
— Très sensible à l'occlusion et aux variations lumineuses de l'image.

1.2.2.7 Apprentissage par Renforcement Profond

Cette méthode [11] considère le processus de recherche d'objets dans l'image comme un processus de décision markovien, ou la boite initiale englobe l'ensemble de l'image et l'agent a plusieurs actions de rétrécissement de l'image possible : réduire (en la soustrayant d'un hyperparamètre alpha) la case dans plusieurs directions possibles et ce jusqu'à converger vers une box entourant l'objet voulu. Et son processus d'apprentissage se base sur une méthode d'apprentissage par renforcement profond.
Avantages :
— La deuxième méthode en qualité de résultat (voir figure 1.6 les lignes "Ours TR" et "Ours AAR") dans la détection de la localisation d'objets aprés les R-CNN mais cependant moins sensible à l'occlusion et troncature que cette dernière.
— Moins sensible à l'occlusion et la troncation que la méthode de R-CNN.
Inconvénients :
— Demande d'hyperparamètrage de la variable α selon la précision voulu concernant la taille des objets à détecter, plus ils sont petits plus petit devra être le paramètre cependant cela augmentera grandement le temps de calcul.
— Méthode rébarbatif car continuant à rechercher des objets dans une image même si ils ont déjà été tous trouvés, et ce jusqu'à l'activation d'un activateur de fin par l'agent.

De par la nature de son processus le calcul du score de cette méthode peut être effectué selon deux modalités possibles :

1.2.2.8 All attended regions (AAR)

Cette modalité prend en compte l'ensemble des régions qui ont été utilisées durant le processus de recherche d'objets dans l'image, c'est une modalité utile que dans le contexte de la méthode qui sera choisie pour ce projet.

1.2.2.9 Terminal regions (TR)

En choisissant cette modalité on opte pour la comparaison avec les régions finales sélectionnées par le modèle, c'est cette modalité qui est utilisé dans l'évaluation de l'ensemble des méthodes proposées (en appliquant un calcul de précision sur les résultats finaux).

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	MAP
DPM [11]	33.2	60.3	10.2	16.1	**27.3**	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
MultiBox [9]	41.3	27.7	30.5	17.6	3.2	45.4	36.2	53.5	6.9	25.6	27.3	46.4	31.2	29.7	37.5	7.4	29.8	21.1	43.6	22.5	29.2
DetNet [32]	29.2	35.2	19.4	16.7	3.7	53.2	50.2	27.2	10.2	34.8	30.2	28.2	46.6	41.7	26.2	10.3	32.8	26.8	39.8	47.0	30.5
Regionlets [38]	44.6	55.6	24.7	23.5	6.3	49.4	51.0	**57.5**	14.3	35.9	45.9	41.3	*61.9*	54.7	44.1	16.0	28.6	**41.7**	*63.2*	44.2	40.2
Ours TR	**57.9**	56.7	38.4	33.0	17.5	51.1	52.7	53.0	17.8	39.1	*47.1*	52.2	58.0	**57.0**	45.2	19.3	42.2	35.5	54.8	49.0	43.9
Ours AAR	55.5	**61.9**	**38.4**	**36.5**	21.4	**56.5**	**58.8**	55.9	**21.4**	**40.4**	46.3	**54.2**	56.9	55.9	**45.7**	**21.1**	**47.1**	41.5	54.7	**51.4**	**46.1**
R-CNN [12]	*64.2*	*69.7*	*50.0*	*41.9*	*32.0*	*62.6*	*71.0*	*60.7*	*32.7*	*58.5*	**46.5**	*56.1*	**60.6**	*66.8*	*54.2*	*31.5*	*52.8*	*48.9*	**57.9**	*64.7*	*54.2*

FIGURE 1.6: Résultats de l'approche par apprentissage par renforcement profond sur le challenge Pascal VOC 2007 © [11]

Chapitre 2

État de l'art de la méthode de détection active d'objets par apprentissage par renforcement profond

La méthode choisie pour ce projet est celle de l'apprentissage profond par renforcement et ce de par l'alliance qu'elle a entre efficacité dans les résultats et résistance aux variations possibles de l'image. Dans ce chapitre les concepts nécessaires à la compréhension de la méthodologie seront introduits.

2.1 Mécanismes de l'apprentissage par renforcement

L'apprentissage par renforcement [12] est une méthode d'apprentissage automatique se concentrant sur le principe d'agents. Dans cette méthode contrairement au reste des techniques d'apprentissage, il n y a pas besoin de connaissance humaine préalable, ainsi le but étant que l'agent apprenne de par ses propre expérimentations dans un modèle sous forme de processus *Markovien*, ou la tâche à accomplir pour chaque agent est de maximiser sa récompense à long terme.

2.1.1 Processus Markovien

Un processus est dit Markovien si dans sa succession d'état la probabilité de transition vers un état suivant s_{t+1} ne dépend que de son état actuel s_t et non pas des états $s_0, s_1, ..., s_{t-1}$, ce qui est donc une propriété d'oubli pouvant être exprimée selon l'équation 2.1

$$\forall t, P(s_{t+1}|s_t) = P(s_{t+1}|s_0, s_1, ..., s_t) \tag{2.1}$$

Un processus de Markov est un tuple (S, P) ou

— S est un ensemble fini d'états.

— P est l'ensemble de la matrice de probabilité de transitions où $P_{s's} = P(s_{t+1} = s'|s_t = s)$.

Dans un processus Markovien on débute donc à un état s_0 et on transite à l'état s_1 selon la probabilité $P_{s_1 s_0}$, puis on suit le même processus pour atteindre l'état s_2 et l'ensemble de ces opérations peuvent être représentées selon la figure 2.2.

$$s_0 \rightarrow^{P_{s_0 s_1}} s_1 \rightarrow^{P_{s_1 s_2}} s_2 \rightarrow^{P_{s_2 s_3}} ... \tag{2.2}$$

2.1.2 Processus de Décision Markovien

Un processus de Décision Markovien (ou *MDP*) est un processus Markovien auquel on introduit les concepts de : récompense, action et un facteur de réduction, il peut être définit comme suit :

— S est un ensemble fini d'états.
— A est l'ensemble fini d'actions.
— P est l'ensemble de la matrice de probabilité de transitions où $P^a_{s's} = P(s_{t+1} = s' | s_t = s, A_t = a)$.
— $\gamma \in [0,1]$ est le facteur de réduction.
— $R : S \times A \to R$ est la fonction de récompense.

Les problèmes en apprentissage par renforcement sont modelés sous forme d'un MDP et tout comme on peut le voir dans la nouvelle définition de P : la probabilité de transiter vers un état ne dépend plus seulement de l'état précédent mais aussi de l'action effectuée/choisie à cet instant t, cette nouvelle représentation suit la figure 2.3.

$$s_0 \to^{a_0} s_1 \to^{a_1} s_2 \to^{a_2} \ldots \tag{2.3}$$

2.1.3 Environnement et Représentation d'état

Un environnement est l'espace dans lequel sont déployés le ou les agents, on peut considérer que l'on a un modèle de l'environnement lorsqu'avec la possession d'un état t, et d'une action, on peut prédire l'état suivant. Un état à un instant s_t est un sous-ensemble extrait de l'environnement, c'est cela que voit l'agent à un moment t et l'information sur laquelle il se basera pour effectuer son prochain choix il peut être résultant de l'environnement tout comme des informations supplémentaires à la prise de décision peuvent y être ajoutés la transformation de l'environnement à un instant t vers une représentation qui sera utilisé par un agent est effectué selon une fonction ϕ .

2.1.4 Actions

Représente toutes les actions que l'agent peut effectuer à un moment donné, par exemple pour un jeu d'échecs cela serait représenté par les déplacements autorisés pour chaque pion du joueur

2.1.5 Récompense

Valeur pouvant être négative ou positive et servant à la convergence du modèle en le poussant à la maximiser sur le long terme

2.1.5.1 Épisode

Un épisode est un ensemble contenant : une série de perceptions de l'agent, ses actions choisies, et les récompenses obtenues le tout l'ayant mené vers une condition de terminaison (attraper une proie par exemple).

2.1.5.2 Fonction de valeur

L'objectif en apprentissage par renforcement est de maximiser l'espérance de retour (ou *"return"*) G_t pouvant être définie selon l'équation 2.4.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0} \gamma^k R_{t+k+1} \tag{2.4}$$

Sachant que le facteur de réduction $\gamma \in [0,1]$ dans cette formule implique que plus on avance dans le futur moins la récompense a de l'importance, plus la valeur de ce facteur est augmenté sa plus on prendra en compte les récompenses futurs. Cela peut être expliqué par plusieurs éléments :

— L'utilisation d'un facteur de réduction permet d'éviter un retour infini dans un MDP circulaire.
— Il existe une incertitude du futur.
— La tendance humaine à favoriser une récompense immédiate plutôt que différée.

Cependant il est toujours possible de garder une importance de récompense constante $\gamma = 1$.

Une politique ou *"policy"* π est une distribution de probabilités des actions sur des états donnés définit par l'équation 2.5. C'est donc cette politique π qui guide les choix d'actions selon état.

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2.5}$$

Il existe deux types de fonctions de valeur qui permettent toutes deux de retrouver la politique optimale, d'abord la **fonction de valeur d'état** définie par l'équation 2.6 et qui représente l'espérance du retour G_t en partant de l'état s et en suivant la politique π.

$$V_\pi(s) = E_\pi[G_t|S_t = s] \tag{2.6}$$

Cette fonction peut être décomposée selon la méthode 2.7

$$\begin{aligned} V_\pi(s) &= E_\pi[G_t|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s] \\ &= E_\pi[R_{t+1} + G_{t+1}|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s] \\ &= \underbrace{E_\pi[R_{t+1}|S_t = s]}_{\text{retour ou récompense immédiate}} + \underbrace{E_\pi[\gamma V_\pi(S_{t+1})|S_t = s]}_{\text{valeur d'état des états suivants}} \end{aligned} \tag{2.7}$$

L'autre fonction est celle représentant la valeur attribuée à une paire État/Action, représentant ainsi l'espérance de retour G_t à obtenir en effectuant une action à un état donné elle est représentée selon l'équation 2.8

$$\begin{aligned} Q_\pi(s,a) &= E_\pi[G_t|S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \end{aligned} \tag{2.8}$$

Si on utilise la définition de l'equation 2.9.

$$R_s^a = E[R_{t+1}|S_t = s, A_t = a] \tag{2.9}$$

On peut aboutir à des équations 2.10 reliant les 2 fonctions de valeurs et la politique π

$$\begin{aligned} V_\pi(s) &= \sum_{a \in A} \pi(a|s) Q_\pi(s,a) \\ Q_\pi(s,a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \end{aligned} \tag{2.10}$$

Et à partir de là, en remplaçant on obtient l'expression de l'équation 2.11 de Bellman qui est lié à la fonction de valeur d'état.

$$V_\pi(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')) \tag{2.11}$$

De plus on peut déduire une équation 2.12 de Bellman pour la fonction de de valeur d'action par état selon l'équation

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a \in A} \pi(a'|s') Q_\pi(s',a')) \tag{2.12}$$

2.1.5.3 Optimisation

Ce qui nous intéresse dans un MDP est de trouver la valeur optimale de la politique π. La fonction de valeur d'état optimale $V_*(s)$ est le maximum de la fonction de valeur d'état sur l'ensemble des politiques possibles comme montré dans l'équation 2.13, elle spécifie ainsi la meilleure performance possible du MDP.

$$V_*(s) = \max_\pi V_\pi(s) \tag{2.13}$$

Il en va de même pour la fonction de valeur d'état par action optimale qui peut être exprimée selon l'équation 2.14

$$Q_*(s,a) = \max_\pi Q_\pi(s,a) \tag{2.14}$$

On définit que la politique optimale π_* doit être meilleure ou égale aux autres politiques possibles, exprimant ainsi la relation que $\forall \pi, \pi_* \geq \pi$.
De ces éléments on arrive [12] au théorème suivant :

- Il existe une politique optimale de sorte que $\forall \pi, \pi_* \geq \pi$.
- Toutes les politiques optimales garantissent en conséquence une fonction de valeur d'état $V_{\pi_*}(s) = V_*(s)$ et une fonction de valeur d'action $Q_{\pi_*}(s,a) = Q_*(s,a)$ par état toutes deux optimales.

En conséquence de ce théorème pour trouver les valeurs optimales de ces fonctions on peut chercher un maximum en suivant les équations 2.15.

$$\begin{aligned} V_*(s) &= \max_a Q_*(s,a) \\ Q_*(s,a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s') \end{aligned} \tag{2.15}$$

Et en effectuant un remplacement on retombe sur des équations 2.16 de Bellman pour l'optimisation qui nous sera utile pour l'algorithme de Q-Learning à venir.

$$\begin{aligned} V_*(s) &= \max_a (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')) \\ Q_*(s,a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} Q_*(s',a') \end{aligned} \tag{2.16}$$

2.1.5.4 Q-Learning

Dans l'algorithme classique de *Q-Learning* afin de stocker l'ensemble des connaissances des associations États/Actions (ou valeurs de la fonction Q) on utilise une matrice que l'on appel "Q-Table" ou matrice Q, ou chaque ligne est un des états connu. Les colonnes représentent toutes les actions possibles, la valeur d'une de ces cases par exemple $Q(etat, action)$ représente la valeur associée à la paire état et action (ou encore l'espérance de récompense). Cette table représente donc la stratégie de l'agent ou "policy" définie précédemment par la variable π. La figure 2.1 montre un exemple de Q-Table ou on peut voir donc son initialisation à des valeurs nulles puis son contenu après plusieurs mise à jour selon l'équation qui va être exposée.

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

FIGURE 2.1: Exemple Q-Table ou les états possibles sont des positions et les actions sont soi des déplacements ou des mouvements d'interactions © Article Wikipédia Q-Learning

Afin de mettre à jour cette Q-Table on fait appel à l'équation de *Bellman* comme montré dans l'équation suivante .

$$\mathrm{Q}^{nouveau}(s_t,a_t) = \underbrace{Q(s_t,a_t)}_{\text{Ancienne valeur}} + \underbrace{\alpha}_{\text{Taux d'apprentissage}} \Big(\underbrace{r_t}_{\text{Récompense accordée à l'instant } t} + \underbrace{\gamma}_{\text{Facteur de réduction}} \underbrace{\max_a Q(s_{t+1},a)}_{\text{Estimation meilleure Q pour l'état successeur } s_{t+1}} - \underbrace{Q(s_t,a_t)}_{\text{Ancienne valeur}} \Big) \quad (2.17)$$

On peut observer dans cette équation qu'après chaque action l'agent met à jour la fonction de valeur en prenant en compte son ancienne valeur et en y ajoutant selon une proportion α la nouvelle récompense (variable à ajuster que l'on peut considérer comme un "taux d'apprentissage") ainsi qu'avec la proportion α ce qu'on appel la différence temporelle ou *"temporal difference"* en Anglais, cette méthode se basant sur le fait que l'on connaisse le modèle (que l'on puisse connaître l'état suivant à partir d'un état et de l'action choisie) et permet ainsi d'apprendre la valeur associée en se basant sur cet état futur, on initialise toutes ces valeurs à 0.

A cette méthode on peut vouloir ajouter de l'aléatoire afin de permettre l'exploration c'est pour cela qu'on ajoute deux paramètres : ε et le *decay rate*, à chaque itération un agent fait un jet aléatoire si celui-ci est inférieur à ε il effectue de l'exploration (en choisissant une action au hasard) sinon il choisit celle ayant la meilleure valeur. Le *decay rate* lui viendra se soustraire d'epsilon à la fin de chaque épisode laissant de moins en moins place à l'exploration. Cette méthode est appelée "ε-Greedy".

La figure 2.2 récapitule la méthode de *Q-Learning* et l'inter-dépendance entre les différents éléments de l'apprentissage par renforcement.

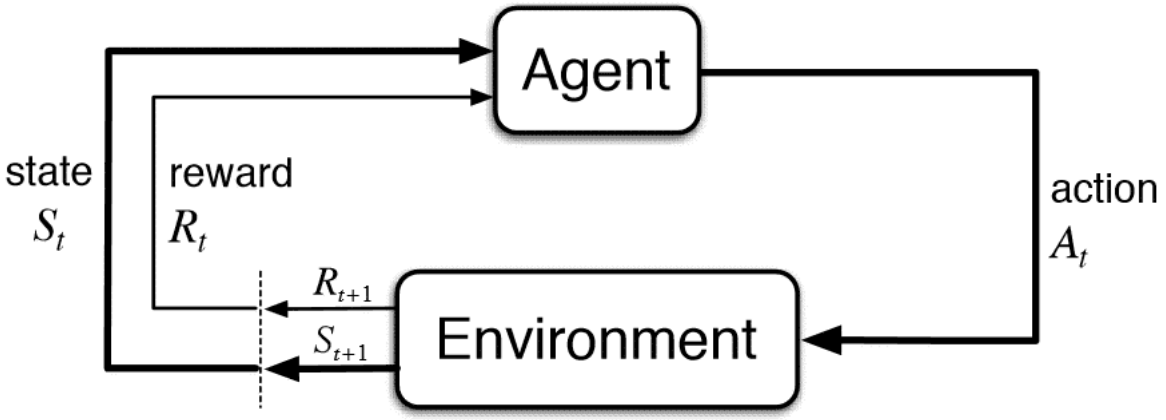


FIGURE 2.2: Algorithme de Q-Learning classique © [12]

2.1.6 Deep-Q-Learning

Le *Deep-Q-learning* se base [13] sur le mélange des techniques d'apprentissage par renforcement et celle des réseaux de neurones profonds (dont plus le nombre de couches est élevé plus ils peuvent apprendre des représentations abstraites) et en considérant un réseau de neurones profond comme fonction d'approximation de l'espérance de la qualité d'une action depuis un état donné et en redéfinissant la policy π comme étant les poids du réseau. Cette technique a fait ses preuves dans la réalisation de tâches complexes (jeu d'échecs, de go ou même des jeux vidéos) en battant les meilleurs joueurs de leurs disciplines. L'objectif étant d'utiliser les réseaux de neurones convolutionnels profonds pour approximer la fonction de valeur d'action optimale exprimée dans l'équation 2.18.

$$Q_*(s,a) = \max_{\pi} E[r_1 + \gamma r_2 + \gamma^2 r_3 + ...|s_t = s, a_t = a, \pi] \tag{2.18}$$

On sait que l'apprentissage par renforcement est connu pour être instable ou même diverger quand une fonction non-linéaire (comme un réseau de neurones) essaie d'approximer la fonction optimale $Q_*(s,a)$. Et cela est causé par : la corrélation entre les séquences observées, le fait que des petites mise à jour de Q peuvent grandement affecter la politique π, l'idée étant de redéfinir la fonction de valeur d'action par état par rapport $Q_*(s,a)$ sous la forme $Q_*(s,a;\theta_i)$ avec θ_i étant les poids du réseaux de neurones convolutionnel.

Une des approches existantes pour résoudre ces problématiques est en se basant sur le *Deep-Q-Learning* et en utilisant le concept de re-jeu d'expériences, approche qui peut être décrite selon l'algorithme 2.

Algorithm 1: Algorithme de Deep-Q-Learning

initialisation de la mémoire de jeu D de capacité N ;
initialisation de la fonction Q avec des poids aléatoires θ ;
initialisation d'une fonction cible $\hat{Q}$ avec les poids $\theta^- = \theta$;
for *Episode 1 à M* **do**
  Initialiser la séquence $s_1 = x_1$;
  for *t = 1 à T* **do**
    Avec une probabilité ε sélectionner une action aléatoire a_t ;
    Sinon selectionner l'action maximisant l'espérance de récompense
    $a_t = \underset{a}{argmax} Q(\phi(s_t), a; \theta)$;
    Executer a_t et récolter la récompense résultante r_t et l'image suivante x_{t+1} ;
    Affecter $s_{t+1} = s_t, a_t, x_{t+1}$ et précalculer $\phi_{t+1} = \phi(s_{t+1})$;
    Stocker les informations liées à la transition $(\phi_t, a_t, r_t, \phi_{t+1})$ dans D ;
    Prendre un echantillon ou "mini-batch" de transitions depuis D ;
    equation
    $$y_j = \begin{cases} r_j, & \text{si l'episode se termine à l'itération j+1} \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta), & \text{sinon} \end{cases}$$
    Effectuer une descente du gradient sur $(y_j - Q(\phi_j, a_j; \theta))^2$;
    Aprés C étapes réinitilialiser $\hat{Q} = Q$;
  end
end

Dans cet algorithme on stocke les expériences passées (actions, récompenses et états liés à chaque itération). Quand un état de terminaison est atteint (l'épisode étant donc terminé) on procède à piocher uniformément des expériences (en effectuant un tel tirage on supprime ainsi le biais de la corrélation entre des tirages successifs qui ne seraient pas aléatoires) et à partir de cet échantillon on met à jour la variable y_j afin d'effectuer une descente du gradient qui nous permet d'optimiser la fonction d'erreur décrite par l'équation 2.19 là ou la valeur minimisée étant celle décrite par l'équation de *Bellman* expliquée précédemment.

$$L_i(\theta_i) = E_{s,a,r}[(E_{s'}[y|s,a] - Q(s,a;\theta_i))^2] \tag{2.19}$$

De plus le réseau de neurones Q est dupliqué en $\hat{Q}$ chaque C itérations et $\hat{Q}$ est utilisé pour générer les cibles y_i cibles. Cette modification a prouvé son augmentation de la stabilité de l'apprentissage du *Deep-Q-Network*.

Ou s est l'état actuel et s' l'état suivant (donc après avoir effectué l'action a et obtenu la récompense r.

La figure 2.3 montre une visualisation des opérations d'un tel réseau de neurones, on peut y voir un réseau de neurones convolutionnel prendre en entrée une image tirée d'un jeu, et avoir en sortie une valeur accordée à chaque action possible du joueur dans cette situation.

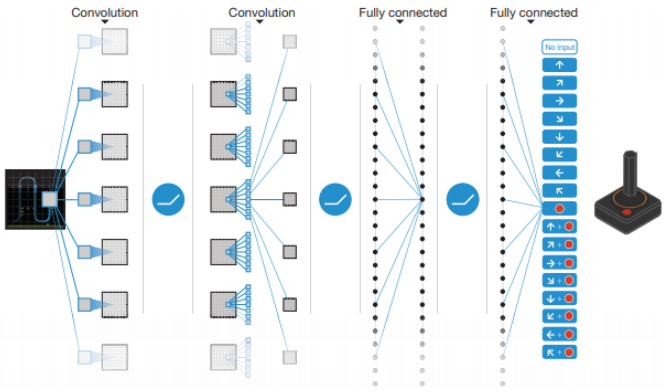


FIGURE 2.3: Visualisation d'Architecture Deep-Q-Learning © [13]

2.2 Réseau de neurones convolutionnel pour l'extraction de caractéristiques

Les réseaux de neurones convolutionnels ou *"CNN"* [14] sont une version spécifique des réseaux de neurones profonds répondant aux besoins liés au contexte de l'image, à savoir que les images n'ont aucune garantie d'invariance, d'être centrées, avoir les même rotations, translations..etc. De plus avec un réseau de neurones classique le concept de voisinnage est ignoré. Le réseau de neurone convolutionnel consiste en l'utilisation de filtres (ou *"kernels"*) et d'appliquer des opérations dites de convolution entre ces derniers et chaque pixel de l'image.

Les réseaux de neurones convolutionnels ont demontré [9] leur efficacité dans les tâches de détection de chiffres manuscrits mais aussi dans la classification d'images et on a pu déterminer (par opération dite de déconvolution) l'interprétation du fonctionnement des filtres, ainsi comme on peut l'observer dans la figure 2.4 à chaque couche dans une architecture de réseaux de neurones

convolutionnel correspond un degré d'abstraction différent : d'abord les contours, puis les textures et des formes de plus en plus concrètes plus on progresse dans les couches. Et les sorties de ces couches sont donc utilisables comme extracteurs de caractéristiques d'images.

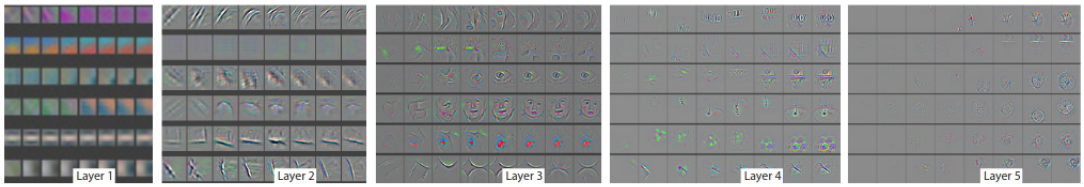


FIGURE 2.4: Filtres par couche d'un réseaux de neurones convolutionnel © [9]

2.3 Normalisation de l'image

Un réseau de neurones convolutionnel prend en entrée une taille fixe, ce qui est problématique dans le contexte d'une *bounding box* à la taille variable, afin de palier à ce problème la méthode utilisée est la même que celle pour la méthode de *R-CNN* [15]. Afin qu'elle soit compatible avec l'architecture de *CNN* (dont l'entrée a une taille fixe de 227×227 pixels). L'image est déformée de manière à atteindre ce format là, par conséquent l'objet y perd certaines proportions mais cela n'a pas un impact important du au fait que le *CNN* est déjà pré-entrainé avec des images déformées. Avant cette déformation l'image est dilatée de manière à ce que la taille de la version déformée soit égale à exactement p pixels de contexte autour de l'image originelle (en choisissant $p = 16$).

Chapitre 3

Modélisation du Problème

Dans ce chapitre la modélisation développée dans le papier sera exposée ainsi que liée aux concepts expliqués dans le chapitre précédent.

3.1 Environnement et Représentation d'état

Dans le contexte de ce papier l'environnement est représenté par la vision actuelle du rectangle de détection ainsi que l'historique des sept actions effectuées précédemment.
On peut voir dans la figure 3.1 l'évolution de cette représentation au fur et à mesure des actions dans une même image.

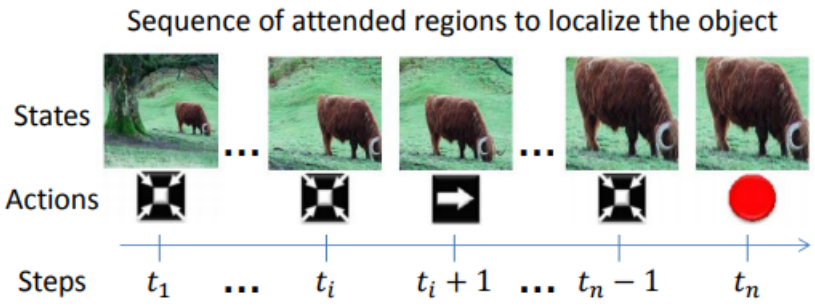


FIGURE 3.1: Représentation d'état © [11]

3.2 Actions

Dans ce papier [11] les actions possibles sont des transformations sur le rectangle de détection en le faisant rétrécir d'un facteur α et cela étant possible dans chaque direction (comme observable dans la figure 3.2), l'action finalisant le processus étant celle de "Trigger" qui valide que tous les objets possibles ont été détectés dans l'image.

FIGURE 3.2: Panoplie d'Actions © [11]

3.3 Récompense

Comme on peut l'observer dans l'équation 3.1 on définit une mesure représentant une progression sur la couverture de la *bounding box* voulu en supposant que b est la zone visée à être découverte et g la zone actuelle de l'agent, cette mesure est ensuite utilisée dans l'équation 4.2 pour l'affectation de la récompense à l'itération t : ainsi si la correspondance est supérieur à un certain facteur τ ajustable l'agent est récompensé, sinon il reçoit un malus (lui permettant de converger non seulement vers la solution mais aussi vers le nombre minimal d'étapes l'y menant).

$$IoU(b,g) = surface(b \cap g)/surface(b \cup g). \tag{3.1}$$

$$R_t = \begin{cases} +\eta, & \text{si } IoU(b,g) \geq \tau \\ -\eta, & \text{sinon} \end{cases} \tag{3.2}$$

3.4 Episode

Dans le contexte de ce papier un épisode débute quand l'agent reçoit une image en entrée et se termine quand l'agent déclare avoir détecté tous les objets de l'image.

3.5 Architecture finale du modèle et procédure d'entraînement

3.5.1 Architecture du modèle

Le modèle final est la combinaison (comme on peut l'observer dans la figure 3.3) de :

— Un réseau de neurone (basé sur une architecture pré-entrainée de réseau de neurones convolutionnels) afin d'extraire les caractéristiques des cadres d'image qu'il reçoit en entrée, mais dont les poids sont figés (il n'apprend donc pas).

— Un réseaux de neurones ou Deep-Q-Network, qui reçoit en entrée les caractéristiques extraites par le premier réseau et effectue une estimation de la valeur de chaque action, puis qui est mise à jour selon la procédure décrite précédemment.

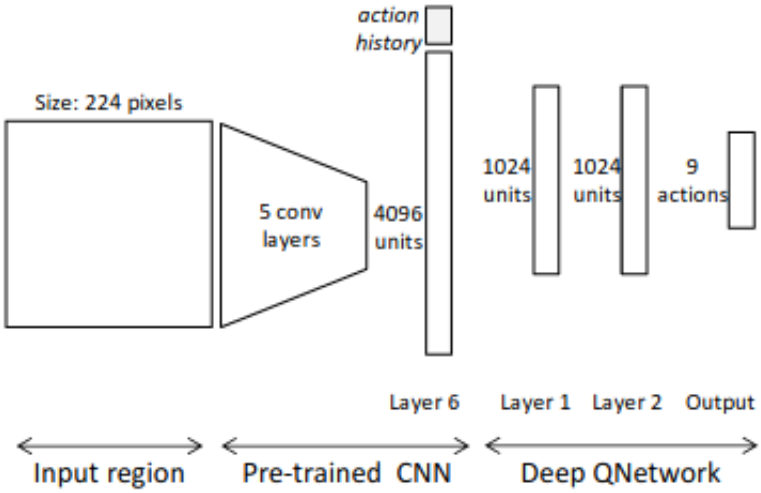


FIGURE 3.3: Architecture Finale © [11]

3.5.2 Processus d'apprentissage

L'objectif de l'agent est de transformer un rectangle en sélectionnant des actions consécutives de manière à maximiser la somme de ses récompenses reçus durant les différents épisodes. L'objectif étant de trouver une stratégie (ou "policy") θ qui définirai l'action à choisir à partir d'un état donné.

La méthode d'entraînement du modèle se déroule comme suit :

— Les paramètres du *Deep-Q-Network* sont initialises aléatoirement.

— L'agent au travers de plusieurs épisodes interagit avec différentes images (en utilisant un méthode "ε-Greedy") afin d'essayer un maximum de scénarios aléatoires et stocker une panoplie d'expériences.

— Une image est prise, transformée selon la méthode de normalisation décrite précédemment.

— Son vecteur de taille $n = 4096$ de caractéristiques est extrait grâce au réseau convolutionnel.

— Ce vecteur est concaténé avec l'historique des actions afin de générer la représentation d'état ϕ.

— La représentation d'état passe entrée du *Deep-Q-Network* et la meilleure action est prédite.

— La récompense est récoltée selon la modalité décrite précédemment, puis le *Deep-Q-Network* est mise à jour selon l'algorithme dédié.

3.6 Jeu de données Pascal et ImageNet

Le jeu de données utilisé pour ce travail est celui de Pascal [16] l'objectif de celui-ci étant de déterminer la performance des méthodes de reconnaissance d'objets sur une large variété d'images naturelles. Dans cet objectif, ce jeu de données contient une large variété d'objets aux variations significatives que ce soit en taille, orientation, pose, luminosité, position et occlusion. Tout en garantissant l'inexistence de biais tel que l'existence d'images centrées et aux conditions favorables . De plus afin d'avoir une évaluation correct des performances, ce jeu de données fournis des images réalistes, aux annotations consistent es et corrects tout en fournissant une bonne variété de situations pour chaque classe donnée.

La figure 3.4 donne quelques exemples pour quelques classes (avion, vélo, oiseau, bateau et bouteille), on peut y voir sur chaque image un ou plusieurs rectangles ou *"bounding box"* chacune contenant une instance de l'objet de la classe en question.

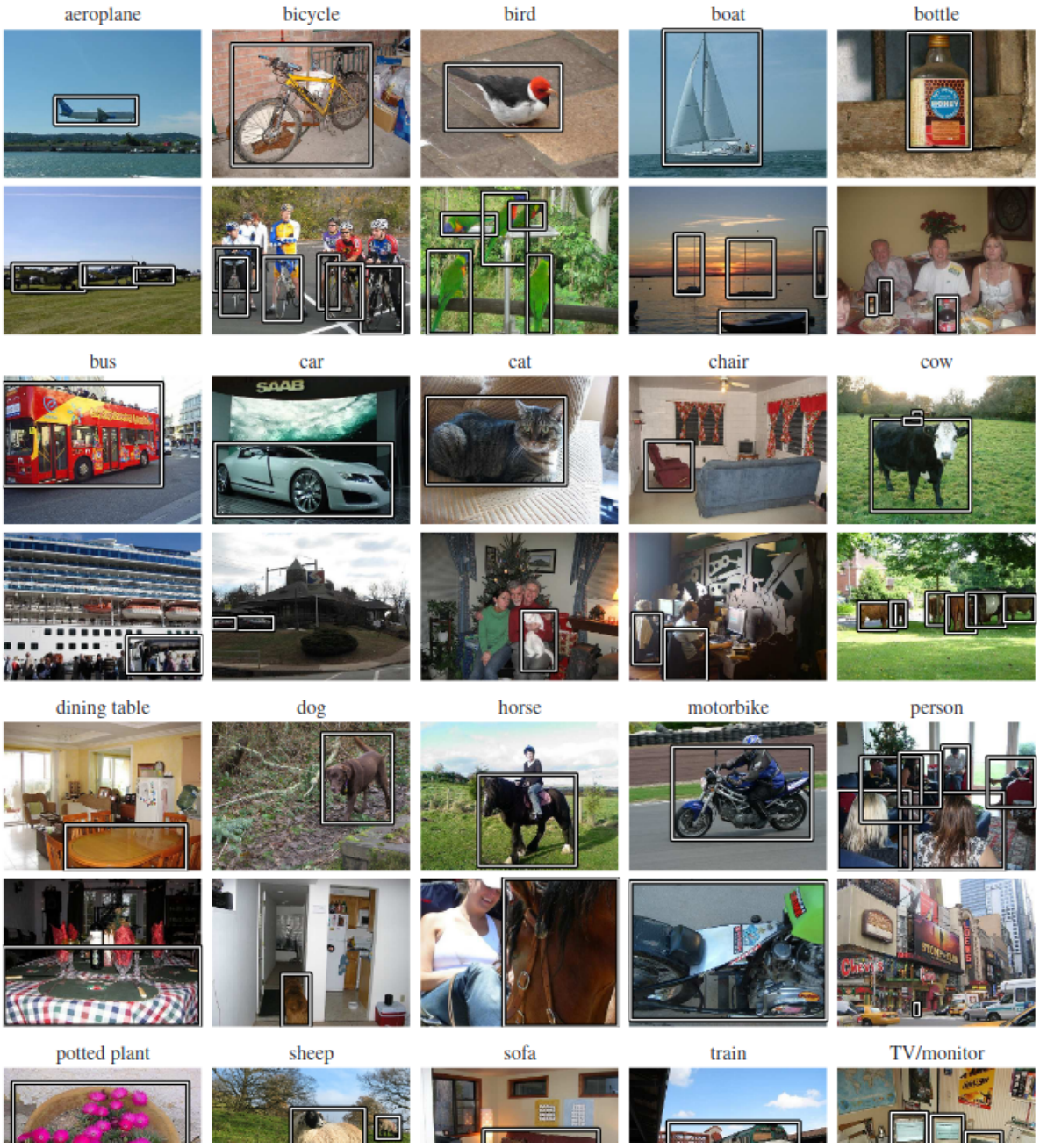


FIGURE 3.4: Jeu de données © [16]

Chapitre 4

Implementation

L'organisation de l'implémentation se divise en plusieurs modules traitant chacun d'une partie différente de la problématique : dataset (jeu de données et pré-traitements), agent (modélisation de la problématique), models (ou sont définies les architectures associées à l'extraction de caractéristiques et au Q-Network), tools (fonctions utiles : affichages des boites englobantes, tri du jeu de données, calcul des différentes métriques).

A partir cela plusieurs on définit plusieurs notebooks jupyter chacun ayant une tâche particulière : entraînement des différentes classes, évaluation de l'efficacité du modèle selon les hyperparamètres, test de l'efficacité sur l'ensemble des classes.

4.1 Jeux de données et pré-traitements

Comme cité dans les sections précédentes le jeu de données utilisé est celui de PASCAL VOC, comme recommandé dans l'article, on utilisera alors le mélange de celui des années 2007 et 2012 afin d'améliorer la qualité de l'apprentissage en ajoutant de la diversité dans les images. Toutes les images du jeu de données sont redimensionnées vers du (224, 224) et passent par une étape de normalisation du contraste, on met à jour ensuite leurs coordonnées de vérité terrain pour les adapter à cette nouvelle taille d'image.

4.2 La classe d'Agent

Cette classe contient toutes les fonctions nécessaires à la gestion des jeux de données et de l'apprentissage, c'est dans cette classe que se situe l'ensemble de la modélisation de la problématique de localisation d'objets.

4.2.1 Actions possibles

A chaque itération les actions possibles sont :

- L'action de "Trigger" qui définit que l'agent pense avoir terminé son processus et ne pas pouvoir faire mieux (en terme d'intersection/union entre la vérité terrain et la prédiction) ce qui fait que l'état actuel est considéré comme un état final.

- Les différentes actions de transformations dépendent du calcul de α_h et α_w qu'on utilise ensuite par de multiples combinaisons d'addition/soustraction aux coordonnées de la boite englobante pour la modifier.

$$\begin{aligned} \alpha_w &= \alpha \times (x_{max} - x_{min}) \\ \alpha_h &= \alpha \times (y_{max} - y_{min}) \end{aligned} \tag{4.1}$$

Cela nous permet donc d'avoir cette panoplie de transformations possibles :

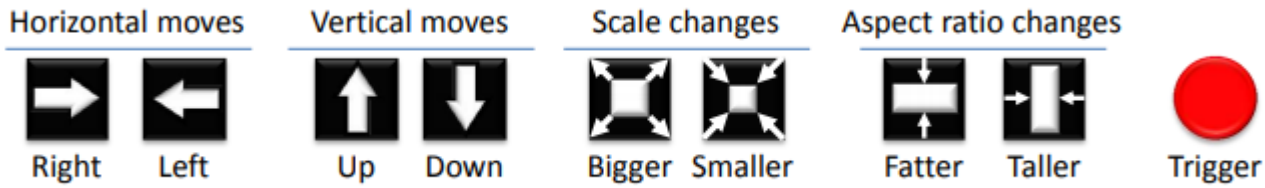


FIGURE 4.1: Panoplie d'actions © [11]

4.2.2 Choix d'action

Le choix des actions se fait en suivant la procédure recommandée dans l'article :

- Une stratégie " ε - greedy" est mise en place c'est à dire ou on définit une variable ε et à chaque itération on fait un jet, si ce jet est inférieur à la valeur d'epsilon on effectue une action aléatoire dite d'exploration (permettant d'explorer de nouvelles options de choix), et si il est supérieur à ε alors on effectue l'action que l'agent considère être la meilleure (dans notre cas l'action qu'aura sélectionné le Q-Network). De plus valeur d'ε décroît avec les itérations laissant de moins en place à l'exploration.

- La partie aléatoire de cette stratégie est remplacée par la mise en place d'un "agent expert" c'est à dire qu'au lieu que l'agent sélectionne des actions aléatoires il choisira une des actions le rapprochant de la vérité terrain (que l'on possède durant l'entraînement).

4.2.3 Système de calcul de récompense

La récompense est calculée selon deux cas possibles :

Récompense d'états finaux

Si on se situe dans un état final on compare notre boite englobante avec celle de la vérité terrain pour obtenir un score d'intersection sur union entre les deux et si ce score est supérieur à un hyper-paramètre τ on récompense le modèle d'un autre hyper-paramètre η.

$$R_t = \begin{cases} +\eta, & \text{si } IoU(prédiction, véritéterrain) \geq \tau \\ -\eta, & \text{sinon} \end{cases} \tag{4.2}$$

Récompense d'états non-finaux
On compare l'intersection sur union de l'état actuel avec la vérité terrain et celui de l'état précédent avec cette même vérité terrain si le score est meilleur on récompense l'agent avec un +1 sinon on lui attribue un malus de -1.

$$R(état, état\ suivant) = signe(\ IoU(état\ suivant, vérité\ terrain) - IoU(état, vérité\ terrain)\ ) \quad (4.3)$$

4.3 Paramétrage de l'algorithme

Le paramétrage de l'algorithme est primordial à l'efficacité de ce dernier, les paramètres importants à prendre en compte sont donc divisés en deux parties :

Apprentissage par Renforcement et modélisation :
On a 3 paramètres importants à prendre en compte :

- η : Va définir la quantité de récompense (ou de malus) attribué à l'état final.
- τ : Définit le seuil comparé à l'intersection/union et en dessous duquel on récompensera l'agent lorsqu'il est à un état final, cela va donc représenter à quel point on veut pousser l'agent à obtenir un résultat final satisfaisant.
- α : Représente la pondération des tailles de transformations effectuées sur l'image, plus ce paramètre est petit moins vite l'agent pourra arriver au résultat ciblé mais plus il pourra être précis dans ce dernier.

Apprentissage profond :
Les paramètres à prendre en compte pour cette partie sont : l'architecture du Q-Network (nombre de couches, probabilités associées à celle de Dropout ..etc.) ainsi que l'architecture utilisée pour l'extraction de caractéristiques (transfer learning).

Dans ce projet on se concentre principalement sur l'évaluation selon les hyper-paramètres liés à la modélisation du problème.

4.3.1 Procédure d'entraînement

L'entraînement se déroule pendant 15 épisodes et ce en suivant l'algorithme suivant :

```
initialisation de la mémoire;
initialisation des images du jeu de données;
initialisation du modèle;
for Épisode 1 à 15 do
    for Image in Images do
        Initialiser l'historique d'actions ( Matrice de zéros (9,9) );
        État = Historique actions + Image;
        État Précédent = État;
        t = 0;
        while Not done do
            État = Historique actions + VGG(Image);
            Action = Choix action(État);
            Image resultante = Transformation(Image, Action);
            t = t + 1;
            if Action == "Trigger" then
                Récompense = Calcul de récompense(Vérité terrain, État);
            else
                Récompense = Calcul de récompense(Vérité terrain, État, État Précédent);
            end
            Ajout Mémoire( ( État Précédent, État, Action, Récompense ) );
            État Précédent = État;
            if t == 40 ou Action == "Trigger" then
                done = Vrai;
            end
        end
        Étape d'optmisation du modèle par Replay memory;
    end
end
```

4.3.2 Procédure d'évaluation

On évaluera les performances du modèle en faisant appel à la fonction d'intersection par union, et on mesurera un seuil sur le jeu de test, si l'intersection/union entre la prédiction du modèle et la vérité terrain de l'image est inférieure à ce seuil on incrémentera le nombre d'éléments bien prédits, et on testera cela avec 5 seuils différents : de 0.1 à 0.5, ce qui permettra ainsi d'observer la précision des boites englobantes prédites par le modèle.

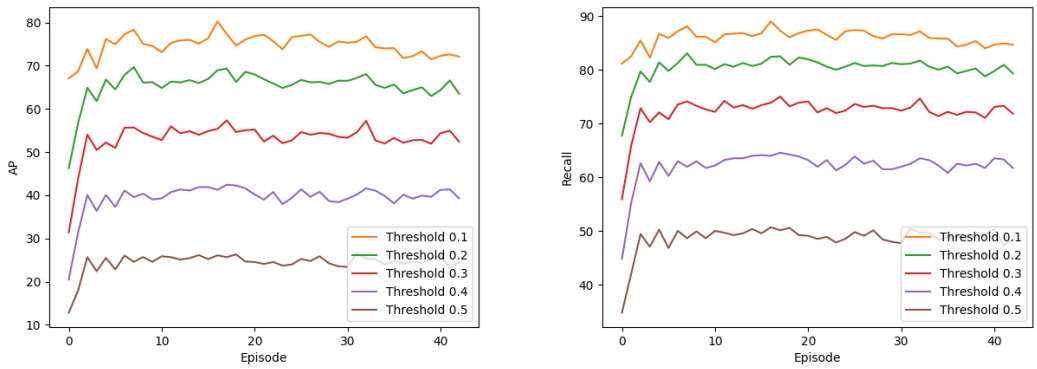
Les aspects testés seront :

— Les différentes combinaisons des hyper-paramètres de la modélisation et leurs résultats sur le jeu de test.
— La performance du modèle sur chaque classe du jeu de test.
— L'évolution des performances du modèle sur le jeu de test de la classe "Dog" après chaque épisode de son entraînement.

Chapitre 5

Resultats

5.1 Evolution sur le jeu d'entraînement



(a) AP (b) Recall

FIGURE 5.1: Evolution pendant l'entraînement

La figure 5.1 représente l'évolution de l'efficacité du modèle durant l'entraînement (mesures d'average précision et recall) à la fin de chaque épisode on effectue une évaluation du modèle sur le jeu de validation. On peut voir sur la figure que le nombre d'épisodes précisé dans l'article c'est à dire 15 épisodes, semble être le moment ou l'efficacité du modèle se stabilise et ce grâce à l'implémentation faite de la technique d'agent expert.

5.2 Résultats sur le jeu de test

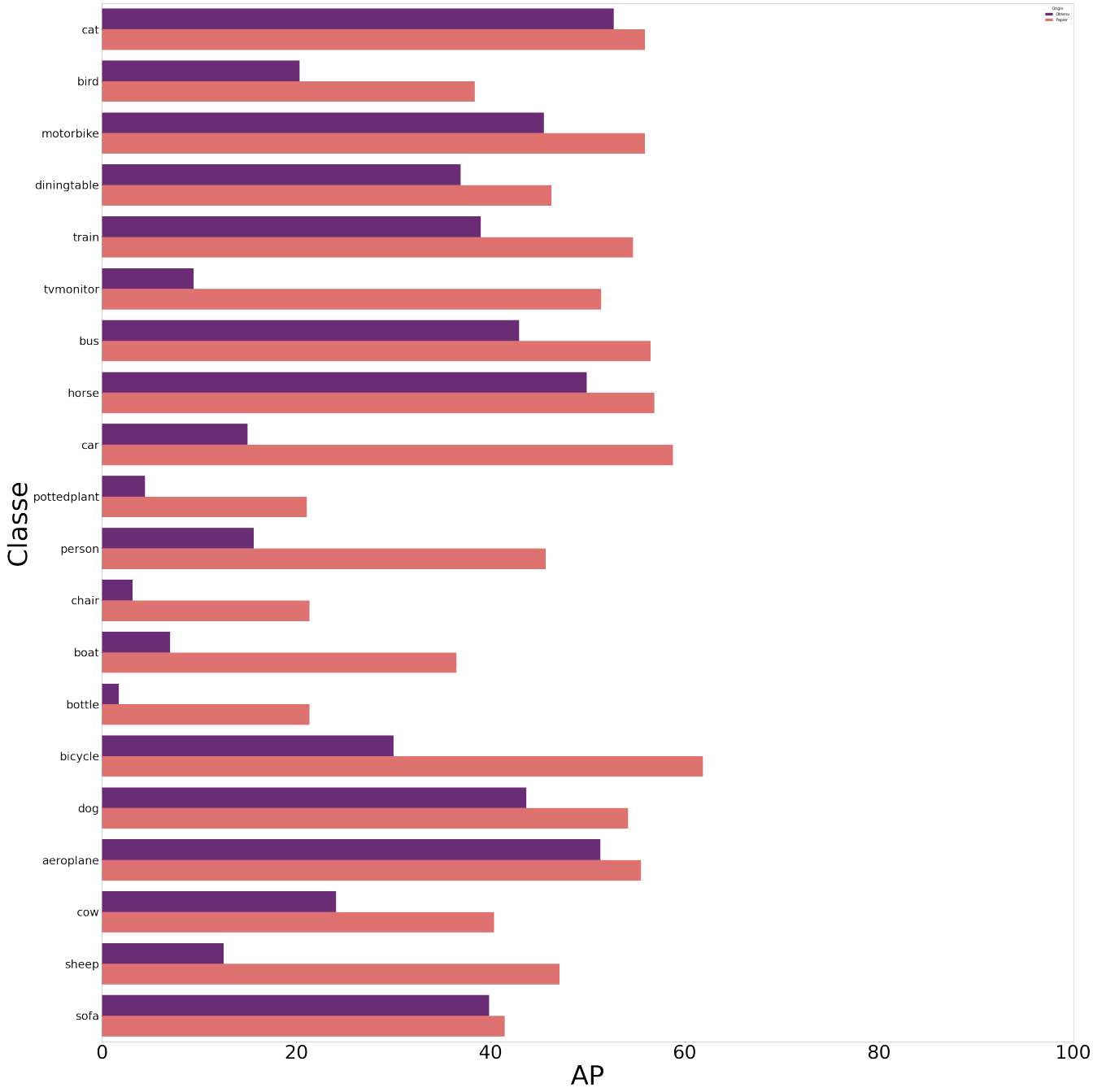
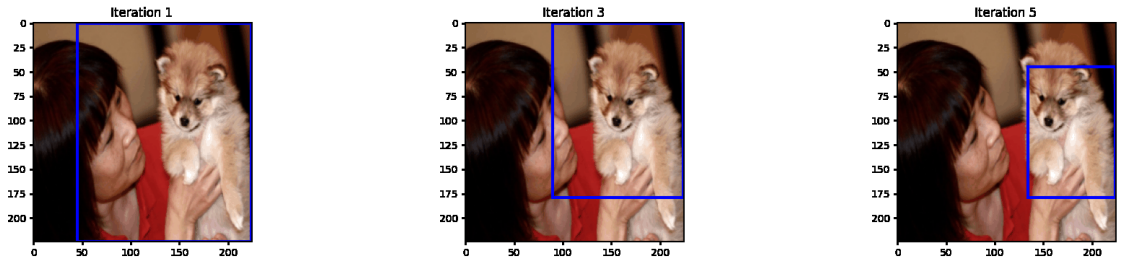


FIGURE 5.2: Résultats sur les classes comparées aux résultats de l'article

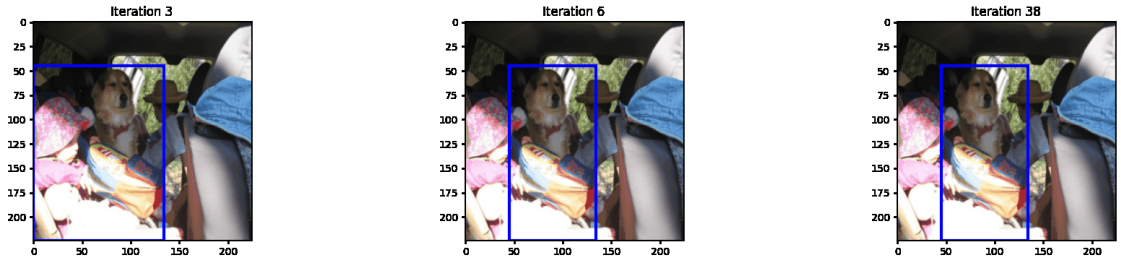
On peut voir sur la figure 5.2 les résultats par classe de notre modèle (représentées par les barres violettes) et celles exposées dans l'article sur le jeu de test. Les résultats obtenus suivent la dynamique générale des performances de l'article, cependant sur certaines classes les résultats sont bien moins bon que ces derniers, cela peut être expliqué par la différence sur le modèle d'extraction de caractéristiques, modèle qui n'est pas précisé dans le papier, la procédure d'entraînement étant assez longue (pouvant prendre jusqu'à trois heures par classe) mon approche a été de tester quelques architectures différentes (vgg-16, googlenet, resnet) et déterminer laquelle donne les meilleurs résultats sur la classe Dog et la garder pour la suite du projet.



(a) Etape 1 (b) Etape 3 (c) Etape 5

FIGURE 5.3: Exemple d'exécution

On voit dans la figure 5.3 un exemple d'une exécution sur une image de chien, on voit que le modèle cible la zone voulue de l'image et effectue un enchaînement d'opérations logiques afin d'englober le chien de manière à optimiser la mesure d'intersection sur union.



(a) Etape 1 (b) Etape 5 (c) Etape 38

FIGURE 5.4: Exemple de mauvaise exécution

Dans l'exemple précèdent on peut voir un type de cas ou le modèle performe mal, on peut voir qu'il localise bien le chien sur la photo cependant il va enchaîner des mouvements d'agrandissement/rétrécissements sur le coté gauche, cela peut être expliqué par le fait que les relations de transformations sont définies par une équation dépendantes des coordonnées actuelles de la boite englobante et que donc plus elle rétrécit plus la transformation se fera d'un facteur plus petit permettant de plus en plus de précision, et c'est cela que semble rechercher l'agent, cependant cette méthode a l'inconvénient d'augmenter grandement le nombre d'itérations qui est alors limité à 40 itérations (selon les recommandations de l'article).

α / η	2	5	10
0.1	52.67	55.44	34.90
0.5	18.28	19.67	18.31
0.8	21.77	21.04	19.13

TABLE 5.1: $\tau = 0.1$

α / η	2	5	10
0.1	52.57	53.23	52.29
0.5	15.15	14.80	11.60
0.8	15.91	18.14	16.65

TABLE 5.2: $\tau = 0.4$

α / η	2	5	10
0.1	52.08	52.53	53.01
0.5	10.65	8.45	9.24
0.8	10.15	9.57	8.31

TABLE 5.3: $\tau = 0.9$

Sur les tableaux précédents on peut voir que parmis toutes les différentes combinaisons d'hyper-paramètres testées, celles ayant donné les meilleures résultats sont celles recommandées par l'article. On peut d'une autre part voir que le paramètre influençant le plus les performances du modèle est α et cela peut être expliqué par le fait qu'augmenter sa valeur impliquerait une grande perte de précision dans les boites englobantes résultantes (car on effectuerait des transformations bien trop conséquentes), il pourrait alors être intéressant de diminuer sa valeur mais il faudrait alors adapter le nombre d'itérations maximum permises au modèle pour qu'il trouve un résultat final.

5.3 Perspectives d'amélioration

— Implémentation d'une méthode de sélection du paramètre α automatique pour l'agent de manière à lui donner la possibilité d'ajuster lui même les transformations d'images qu'il effectue et ainsi ne pas avoir à répéter plusieurs fois une même action.
— Test de différentes architectures de Q-Network et de réseaux d'extraction de caractéristiques et observer lesquelles amélioreront les résultats globalement sur les classes.
— Ajouter des possibilités d'actions plus exhaustives permettant une convergence plus rapide (mais complexifiant la procédure d'apprentissage pour l'agent).

Conclusion

En utilisant les méthodes d'apprentissage par renforcement et la modélisation par modèle markovien on peut résoudre une variété de problématiques assez diverses, et comme on a pu le constater durant cette implémentation on peut même s'attaquer à la problématique de localisation d'objets.

Des résultats très satisfaisants ont ainsi pu être observées sur plusieurs classes différentes même si que ce soi de par les résultats de l'article ou ceux obtenus durant l'implémentation on peut voir que cette méthode est assez dépendante de la quantité de données et pas forcément généralisable à toutes les classes d'images efficacement.

Plusieurs aspects dans la modélisation de l'article restent explorables : augmenter le nombre d'itérations maximum et diminuer la valeur d'α, augmenter le nombre de données d'entraînement en ajoutant celles d'autres dataset de localisation d'objets, l'ajout de nouvelles actions ou possibilités d'interactions de l'agent avec ses propre hyper-paramètres.

On peut aussi alors ouvrir la modélisation de manière à traiter de la problématique de segmentation d'images et ce soi en adaptant la modélisation de l'article, ou alors en l'utilisant couplé à des algorithmes de visualisation tel que GradCAM pour extraire des cartes de segmentation.

Enfin, cette implémentation m'a permis de noter qu'il faut être vigilant, car même si on guide l'agent lors de son apprentissage au travers de mécanismes tels que les récompenses ou l'implémentation d'agent expert, des comportements imprévus peuvent être appris et il faut alors adapter notre modélisation de manière à les faire disparaître.

Bibliographie

[1] Felzenszwalb, Pedro Girshick, Ross Mcallester, David Ramanan, Deva. (2010). **"Object Detection with Discriminatively Trained Part-Based Models"**. IEEE transactions on pattern analysis and machine intelligence. 32. 1627-45. 10.1109/TPAMI.2009.167.

[2] Lampert, Christoph Blaschko, Matthew Hofmann, Thomas. (2008). **"Beyond sliding windows : Object localization by efficient subwindow search"**. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 1-8 (2008). 1 - 8. 10.1109/CVPR.2008.4587586

[3] Malisiewicz, Tomasz Mulam, Harikrishna Efros, Alexei. (2011). **"Ensemble of exemplar-SVMs for object detection and beyond"**. Proceedings of the IEEE International Conference on Computer Vision. 89-96. 10.1109/ICCV.2011.6126229.

[4] Ren, Shaoqing and He, Kaiming Girshick, Ross Sun, Jian. (2015). **"Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks"**. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

[5] Girshick, Ross. (2015). **"Fast r-cnn"**. 10.1109/ICCV.2015.169.

[6] Ren, Shaoqing He, Kaiming Girshick, Ross Sun, Jian. (2015). **"Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks"**. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

[7] Xu, Kelvin Ba, Jimmy Kiros, Ryan Cho, Kyunghyun Courville, Aaron Salakhutdinov, Ruslan Zemel, Richard Bengio, Y.. (2015). **"Show, Attend and Tell : Neural Image Caption Generation with Visual Attention"**. arXiv :1502.03044

[8] Sermanet, Pierre Eigen, David and Zhang, Xiang and Mathieu, Michael and Fergus, Rob and Lecun, Yann. (2013). **"OverFeat : Integrated Recognition, Localization and Detection using Convolutional Networks"**. In International Conference on Learning Representations (ICLR) (Banff).

[9] Sermanet, Pierre Eigen, David Zhang, Xiang Mathieu, Michael Fergus, Rob Lecun, Yann. (2013). **"OverFeat : Integrated Recognition, Localization and Detection using Convolutional Networks"**. International Conference on Learning Representations (ICLR) (Banff).

[10] Hoiem, Derek Chodpathumwan, Yodsawalai Dai, Qieyun. (2012). **"Diagnosing Error in Object Detectors"**. 340-353. 10.1007/978-3-642-33712-3_25.

[11] Caicedo, Juan Lazebnik, Svetlana. (2015). **"Active Object Localization with Deep Reinforcement Learning"**. 10.1109/ICCV.2015.286.

[12] Sutton, Richard Barto, Andrew. (1998). **"Reinforcement Learning : An Introduction"**. IEEE transactions on neural networks/ a publication of the IEEE Neural Networks Council. 9. 1054. 10.1109/TNN.1998.712192.

[13] Mnih, Volodymyr Kavukcuoglu, Koray Silver, David Rusu, Andrei Veness, Joel Bellemare, Marc Graves, Alex Riedmiller, Martin Fidjeland, Andreas Ostrovski, Georg Petersen, Stig Beattie, Charles Sadik, Amir Antonoglou, Ioannis King, Helen Kumaran, Dharshan Wierstra, Daan Legg, Shane Hassabis, Demis. (2015). **"Human-level control through deep reinforcement learning"**. Nature. 518. 529-33. 10.1038/nature14236

[14] Lecun, Yann Haffner, Patrick Bengio, Y.. (2000). **"Object Recognition with Gradient-Based Learning"**.

[15] Girshick, Ross Donahue, Jeff Darrell, Trevor Malik, Jitendra. (2014). **"Rich feature hierarchies for accurate object detection and semantic segmentation"**.

[16] Everingham, Mark Eslami, S. Van Gool, Luc Williams, Christopher Winn, John Zisserman, Andrew. (2014). **"The Pascal Visual Object Classes Challenge : A Retrospective"**. International Journal of Computer Vision. 111. 10.1007/s11263-014-0733-5.