Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM / ICAR**

Spécialité: **Informatique**

Présentée par **Jérôme Pasquet**
pasquet@lirmm.fr

Modélisation, détection et classification d'objets urbains à partir d'images photographiques aériennes

Soutenue le 03/11/2016 devant le jury composé de

Rapporteurs		
M. Vincent CHARVILLAT	Pr	IRIT, ENSEEIHT Toulouse
M. Christophe GARCIA	Pr	CNRS, INSA, LIRIS Lyon
Examinateurs		
M. Matthieu CORD	Pr	CNRS, LIP6 Paris
M. Pascal PONCELET	Pr	CNRS, LIRMM Montpellier
Invité		
M. Laurent DERUELLE	MCF	LRA - Berger-Levrault
Directeur de thèse		
M. Marc CHAUMONT	MCF	Université de Nîmes - LIRMM Montpellier
Co-encadrant		
M. Gérard SUBSOL	CR	CNRS, LIRMM Montpellier

UNIVERSITÉ DE MONTPELLIER

Remerciements

Dans un premier temps, je remercie Mustapha Derras, Laurent Deruelle et Francois Bibonne (la première année) de la société Berger-Levrault de m'avoir accordé leur confiance et de m'avoir permis d'effectuer ma thèse dans de bonnes conditions. De plus, je remercie Berger-Levrault, qui a fourni un financement afin que je puisse participer à des conférences, workshops et écoles sans aucune difficulté.

Dans un second temps, je remercie les professeurs Vincent Charvillat et Christophe Garcia d'avoir accepté d'être rapporteurs de ma thèse. Je remercie également le professeur Matthieu Cord d'avoir été examinateur de ma thèse et pour ses propositions d'améliorations du manuscrit très pertinentes. Afin je remercie le professeur Pascal Poncelet d'avoir présidé mon jury de thèse.

Dans un troisième temps, je tiens à remercier mon directeur de thèse Marc Chaumont et co-encadrant de thèse Gérard Subsol de m'avoir enseigné le métier de chercheur. Plus particulièrement je remercie Marc de m'avoir montré par le biais de corrections multiples la rigueur à avoir dans les notations mathématiques et dans la rédaction de papiers. De la même manière, merci à Gérard de m'avoir montré et d'avoir repris (de très nombreuses fois) mes titres afin de les rendre toujours plus explicites.

Pour poursuivre je ne manquerai pas de remercier l'équipe ICAR dans sa totalité pour les activités proposées, les légendaires C&C et l'ambiance quotidienne. Évidemment je remercie particulièrement William qui a fermé les yeux sur mon refus de saluer l'ensemble des membres de l'équipe chaque matin (mais je pense avoir progressé sur ma sociabilité durant ces 3 ans !). Merci également à Nicolas, Laurie et Guylène et à leurs temps de réponse ultra rapide. Enfin, un double merci également aux joueurs participant aux soirées jeux de sociétés qui étaient, selon moi, trop peu nombreuses.

Je garde bien sur une pensée pour mes camarades et amis de licence, master et thèse. Merci à Axel de l'IES de m'avoir fait connaître la légende de la Fondation. Merci à Jean-Michel pour le running. Merci à Julien et Mathilde d'être partis à Toulouse, ce qui diminua considérablement mon temps de jeu et donc augmenta mon temps de travail. Merci à mes autres amis de longues dates de ne pas vous vexer, mais les places dans les remerciements sont limitées.

Afin je remercie ma famille pour son soutien. Ma petite soeur d'avoir raté sa première année de médecine me permettant d'être docteur avant elle. Ma grande soeur d'être ma grande soeur. Mes parents pour leur aide quand j'en ai besoin et Mèmère pour la relecture (douloureuse) de mon manuscrit. Évidemment je remercie ma belle famille de manière similaire.

Pour finir, je m'excuse auprès de Pony à qui aucun remerciement écrit ou oral ne pourra jamais représenter ma gratitude. Merci d'exister.

Abstract

This thesis deals with the problems of automatic localization and recognition of urban objects in high-definition aerial images. Urban object detection is a challenging problem because they vary in appearance, color and size. Moreover, there are many urban objects which can be very close to each other in an image. The localization and the automatic recognition of different urban objects, considering these characteristics, are very difficult to detect and classical image processing algorithms do not lead to good performances. We propose then to use the supervised learning approach. In a first time, we have built a Support Vector Machine (SVM) network to merge different resolutions in an efficient way. However, this method highly increases the computational cost. We then proposed to use an "activation path" which reduces the complexity without any loss of efficiency. This path activates sequentially the network and stops the exploration when an urban object has a high probability of detection. In the case of localizations based on a feature extraction step followed by a classification step, this may reduce by a factor 5 the computational cost. Thereafter, we show that we can combine an SVM network with feature maps which have been extracted by a Convolutional Neural Network. Such an architecture associated with the activation path increased the performance by 8% on our database while giving a theoretical reduction of the computational costs up to 97%. We implemented all these new methods in order to be integrated in the software framework of Berger-Levrault company, to improve land registry for local communities.

keywords: *aerial image processing, Supervised learning, Deep learning, urban object localization*

Résumé

Cette thèse aborde des problèmes liés à la localisation et reconnaissance d'objets urbains dans des images aériennes de très haute définition. Les objets urbains se caractérisent par une représentation très variable en terme de forme, texture et couleur. De plus, ils sont présents de multiples fois sur les images à analyser et peuvent être collés les uns aux autres. Pour effectuer la localisation et reconnaissance automatiquement des différents objets nous proposons d'utiliser des approches d'apprentissage supervisé. De part leurs caractéristiques, les objets urbains sont difficilement détectables et les approches classiques de détections n'offrent pas de performances satisfaisantes. Nous avons proposé l'utilisation d'un réseau de séparateurs à vaste marge (SVM) afin de mieux fusionner les informations issues des différentes résolutions et donc d'améliorer la représentativité de l'objet urbain. L'utilisation de réseau de SVM permet d'améliorer les performances mais à un coût calculatoire important. Nous avons alors proposé d'utiliser un chemin d'activation permettant de réduire la complexité sans perdre en efficacité. Ce chemin va activer le réseau de manière séquentielle et stoppera l'exploration lorsque la probabilité de détection d'un objet est importante. Dans le cas d'une localisation basée sur l'extraction de caractéristiques puis la classification, la réduction calculatoire est d'un facteur cinq. Par la suite, nous avons montré que nous pouvons combiner le réseau de SVM avec les cartes de caractéristiques issues de réseaux de neurones convolutifs. Cette architecture combinée avec le chemin d'activation permet une réduction théorique du coût d'activation pouvant aller jusqu'à 97% avec un gain de performances d'environ 8% sur les données utilisées. Les méthodes développées ont pour objectif d'être intégrées dans un logiciel de la société Berger-Levrault afin de faciliter et d'améliorer la gestion de cadastre dans les collectivités locales.

mots-clefs : *Traitement images aériennes, Classification supervisée, Apprentissage profond, Localisation d'objets*

Table des matières

Chapitre 1

Introduction

1.1 Contexte de la thèse

Les travaux de cette thèse ont été réalisés dans le cadre d'un contrat CIFRE entre la société Berger-Levrault et l'équipe-projet ICAR du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM).

La société Berger-Levrault est experte du droit public des administrations publiques dans les domaines de la santé, du sanitaire, du social, et de la gestion de territoires. Son activité est à 87% consacrée à l'édition de logiciel. Dans le cadre de la gestion de territoires, Berger-Levrault met notamment à la disposition des collectivités locales une application de gestion des cimetières, nommée E-cimetière [1].

Cette application propose un ensemble de solutions clef-en-main pour l'administration des cimetières. Elle permet notamment le suivi et la gestion de concessions, l'historique des travaux et des interventions, la gestion de factures... De plus, elle propose une cartographie des cimetières en géolocalisant l'ensemble des tombes et des concessions. Cela facilite la navigation dans le cimetière et offre donc une meilleure gestion de celui-ci. Dans le futur, elle pourrait rattacher la position des concessions à leurs propriétaires afin de faciliter la localisation par des visiteurs. Actuellement, la géolocalisation des tombes dans les cimetières est réalisée par un expert géomètre devant se déplacer et effectuer les relevés sur le terrain. Cependant ce processus de cartographie est très coûteux en temps et en ressources.

Une des difficultés est de localiser automatiquement et précisément les concessions dans les cimetières. Une solution consiste à rechercher ces objets dans des images aériennes. Pour effectuer cette tâche de traitement de l'image, Berger-Levrault s'est rapproché de l'équipe-projet ICAR en 2011. À la suite de cette première collaboration deux stages ont été financés en 2011 et 2012. Ils ont permis de tester des méthodes connues de la littérature à savoir l'approche de Viola et Jones [116] et l'approche de Aldavert et al. [3]. Il s'est finalement avéré

1. http://www.berger-levrault.com/solutions/fonction-publique-territoriale/services-citoyens/e-cimetiere.html

que le problème de la localisation des tombes dans les cimetières n'était pas simple et qu'il impliquait un investissement en recherche sur le long terme. C'est dans ce contexte que cette thèse a débuté en 2013. Pour autant, cette thèse traite de problèmes plus larges de détection d'objets urbains que nous définirons dans la prochaine section. Il est à noter qu'un critère important est de permettre à moyen terme une industrialisation des algorithmes développés dans nos recherches. Cela implique que nous devons prendre soin d'optimiser les coûts de calculs.

1.2 Présentation des données

Afin de réaliser ces travaux de thèse, la société Berger-Levault a mis à disposition une collection d'images aériennes en couleur de très haute définition de cimetières de villes et villages français. La résolution au sol varie de 2.5 cm à 5 cm par pixel et la couleur est codée sur 3 canaux (rouge, vert et bleu) de 8 bits. Pour une zone donnée, plusieurs images sont acquises, puis elles sont orthorectifiées et mises en mosaïques afin de former une image continue. La taille des cimetières varie de 5000×5000 à 11000×11000 pixels. Ces cimetières proviennent de villages et villes, majoritairement de Haute-Marne, avec des tailles différentes et peuvent contenir quelques dizaines de tombes à plusieurs milliers (cf figure 1.1).

FIGURE 1.1 – Exemple d'un cimetière contenant une trentaine de tombes à gauche et d'un cimetière avec plus de mille tombes à droite.

Les photos présentées dans la figure 1.1 mettent en avant la taille variable des volumes à traiter. La base totale possède 30 images, dont l'ensemble contient environ 9 200 tombes. Le tableau 1.1 donne la liste complète des noms

des villes et du nombre de concessions visibles où les photographies aériennes ont été acquises.

Nom	Nb concessions	Nom	Nb concessions	Nom	Nb concessions
Balesmes	71	Marines	508	Rolampont	176
Beuvry Montage	1359	Humes Jorquenay	48	Saint-Ciergues	97
Bourg	51	Jorquenay	91	Saint-Geosmes	215
Champigny	132	La maye Malherbe	547	Saint-Martin	48
Chanoy	38	Lannes	115	Saint-Maurice	65
Charmoilles	96	Marquillies	317	Saint Saulve	77
Cosne cours sur Loire	330	Mortagne	698	Vieux Moulin	45
Courbert	178	Noidant le rocheux	104	Voisines	94
Gandrange	58	Perrancey	64	Lecey	95
Litaldus	1309	Folembray	790	Arrou	1441

TABLE 1.1 – Liste des villes et villages dont les images de cimetières sont acquises, ainsi que le nombre de concessions observables.

1.3 Caractérisation d'un objet urbain

Dans le cadre de cette thèse, nous allons définir ce que nous considérons comme étant un objet urbain. Pour cela, nous dressons ci-après une liste de caractéristiques qui définit les objets urbains :

— Un objet urbain possède une **taille variable** et **aucune direction préférentielle**. Contrairement aux applications telle que la détection de visages dans des photos, où les têtes sont souvent droites ou légèrement inclinées, les objets urbains peuvent être orientés selon toutes les directions. Dans le cadre des tombes, nous supposons que l'orientation est inconnue, d'autant plus que certains cimetières ont des organisations circulaires, comme le montre la figure 1.2. A cette difficulté réelle, s'ajoute une problématique plus classique à savoir, la résolution de l'image qui n'est pas la même d'une image à une autre.

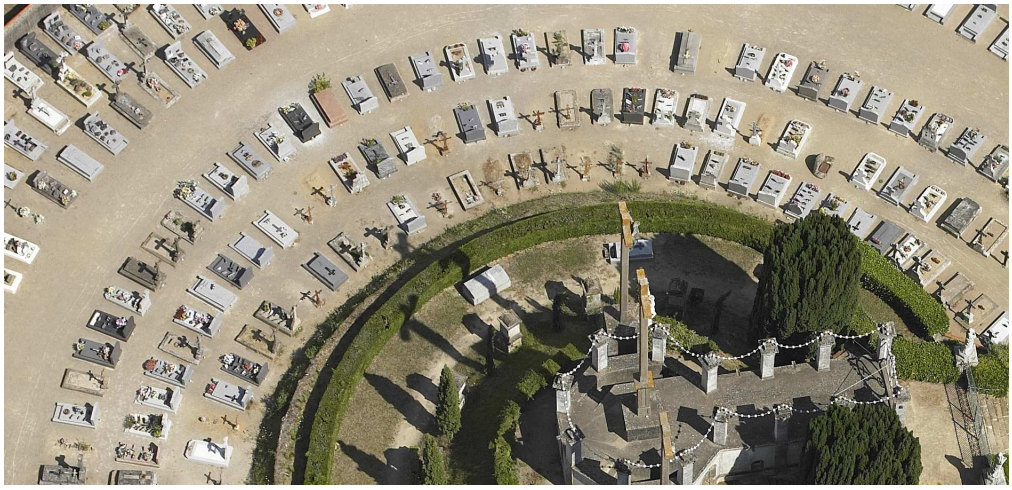


FIGURE 1.2 – Exemple d'un cimetière avec une organisation circulaire.

— Un objet urbain de par sa localisation est soumis à des occultations partielles et à la **présence d'ombres**. Dans le cadre des cimetières, le phénomène d'ombres est très présent à cause des grands arbres (souvent des cyprès) qui sont traditionnellement plantés à proximité des tombes, ainsi que la présence de chapelle dans le cas de villages. La photo de gauche sur la figure 1.1 permet de voir le phénomène d'ombres. Sur la photo de droite dans la figure 1.1, on remarque que certains arbres recouvrent totalement et partiellement la vue aérienne sur des tombes les rendant plus difficilement détectables.
— Un objet urbain possède de nombreuses "représentations" au sens géométrique et textuel ce qui le rend **difficilement caractérisable**. La figure 1.3 illustre quelques variations possibles d'une tombe à une autre. Dans la catégorie A, on distingue des tombes simples avec une dalle et peu d'objets posés dessus alors que d'autres sont ornées de nombreux objets, notamment des plaques. Dans la catégorie B on constate la présence d'une forte concentration de végétaux. Or, en fonction de la saison et de l'entretien des tombes, ces végétaux ont une texture et une couleur très variables. La catégorie C met en évidence des tombes sans contour net qui peuvent se confondre avec le sol. La catégorie D représente des tombes dont la forme est atypique, par exemple une tombe hexagonale ou très haute. L'ensemble de ces catégories nous permet d'affirmer que les tombes ont une grande variabilité, ce qui les rend difficilement reconnaissables.

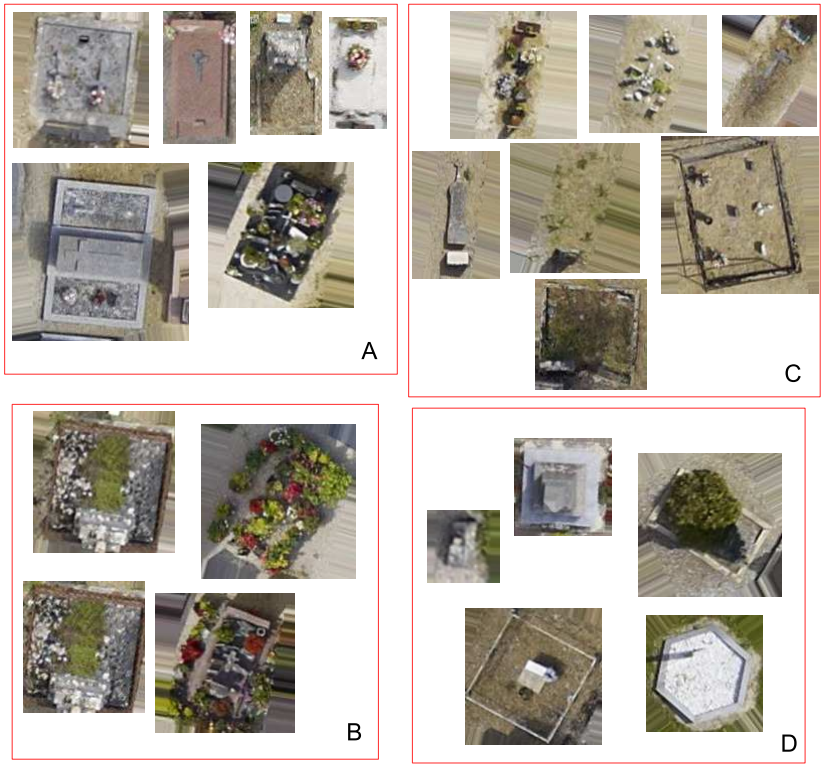


FIGURE 1.3 – Exemple de quelques tombes rangées en quatre catégories. La catégorie A montre des tombes classiques avec une dalle et une croix. La catégorie B expose des tombes possédant beaucoup de végétation. La catégorie C est composée de tombes sans dalle et avec un contour parfois inexistant. Enfin, la catégorie D comporte des tombes avec des formes variées, telles que des formes hexagonales ou avec un fort relief.

— Les objets urbains sont **multiples** et peuvent être **collés les uns aux autres**. La distance entre deux tombes est très variable. Pour une séparation de l'ordre de 5 pixels ou plus, il semble assez simple de détecter la présence de contours (s'ils existent, voir figure 1.3) et donc de localiser la tombe. Concrètement beaucoup de tombes ne sont séparées par aucune frontière franche, ce qui en fait un cas d'étude sur la détection d'objets, peu étudié dans la littérature. Sur la photo de gauche de la figure 1.4, nous montrons qu'il est difficile de définir la frontière entre la tombe avec l'accolade jaune et celle avec l'accolage rouge. De plus, aucune supposition sur les tailles des tombes ne peut être faite puisqu'elles varient entre 45 et 130 pixels en largeur. Enfin, comme le montre la photo de droite sur la figure 1.4 et la figure 1.2 il est difficile de déterminer une règle d'alignement dans les cimetières même quand les tombes sont proches.

FIGURE 1.4 – Dans le cimetière de gauche, les trois accolades montrent la présence de trois tombes difficilement localisables car les frontières ne sont pas clairement définies. Dans le cimetière de droite, nous constatons que l'alignement des tombes n'est pas trivial malgré une forte proximité entre elles.

1.4 Bilan des difficultés du problème

L'objectif de la thèse est la détection et la localisation d'objets urbains dans des images aériennes. Nous nous sommes particulièrement intéressés à la détection de tombes. Ce type d'objets possède de grandes variabilités de tailles, d'orientations, de textures, de couleurs et de voisinages à proximité. Une des caractéristiques particulières des tombes est qu'elles peuvent être collées les unes aux autres. L'ensemble de ces critères en fait un cas d'étude rare.

Afin de localiser les tombes, une première solution simple est de détecter des rectangles dans les images. Cependant, comme nous l'avons mentionné dans la section précédente, la tombe n'a pas toujours de contours prononcés. Nous avons dirigé nos travaux de recherche vers une détection par apprentissage supervisé. Il s'agit de méthodes d'apprentissage où l'on cherche à modéliser un objet par le biais d'exemples d'apprentissage. Afin d'obtenir cette base d'apprentissage il est nécessaire d'effectuer une annotation manuelle de la position de l'ensemble des tombes. Ce travail fastidieux a été effectué, à l'aide d'un outil graphique que nous avons réalisé et à la contribution de plusieurs stagiaires de L1 encadrés durant cette thèse.

1.5 Plan

La suite du manuscrit est composée d'une première partie sur l 'état de l'art et d'une seconde partie consacrée à nos contributions.

Dans la première partie, nous définirons dans le chapitre 2, la description d'une image et la construction des descripteurs associés. Puis, nous aborderons dans le chapitre 3 l'identification et la localisation des objets dans des images via des méthodes d'apprentissage automatique. Pour cela, nous détaillerons et comparerons deux familles, à savoir celle des approches supervisées par pixels et celle des approches supervisées par objets. Nous terminerons cet état de l'art par les approches d'apprentissage profond dans le chapitre 4.

Dans la seconde partie, nous décrirons et évaluerons nos contributions. Le chapitre 5 détaillera les méthodes et métriques utilisées pour l'évaluation de nos algorithmes. Puis, dans le chapitre 6 nous mettrons en évidence un problème de représentation des caractéristiques utilisées, pour lequel nous proposerons une solution basée sur l'utilisation d'un réseau de SVM augmentant significativement les performances. Cependant, ce réseau amplifie considérablement les temps de calculs. Dans le chapitre 7, nous mettrons en place un chemin d'activation permettant de réduire ce coût de façon drastique. Puis, dans le chapitre 8, nous adapterons un réseau de convolutions en utilisant un réseau de SVM et un chemin d'activation afin de gagner en efficacité et de réduire la complexité.

Finalement, nous conclurons le manuscrit dans le chapitre 9, où nous résumerons les contributions apportées et nous proposerons plusieurs pistes d'améliorations sur un court et un long terme.

Première partie

État de l'art

Chapitre 2

Extraction de caractéristiques

2.1 Introduction

Dans cette thèse, nous cherchons à reconnaître et à localiser des objets urbains et tout particulièrement des tombes dans des images aériennes. Nous représentons les images couleur par une matrice 3D notée $I_{x,y,z}$, avec x, y les coordonnées du pixel sur l'image et z le canal couleur utilisé. Dans notre cas, nous utiliserons l'espace couleur rouge, vert et bleu pour l'encodage des images aériennes, et donc $z \in \{r, v, b\}$.

Les images peuvent se décrire de deux façons différentes. La première description est globale et consiste à caractériser la totalité de l'image. Cela permet en particulier d'indexer et de retrouver des images dans des bases de données [20, 56]. Le second type de description est local et consiste à extraire un ensemble de descripteurs dans des sous-parties de l'image. Cet ensemble de descripteurs locaux peut également être utilisé pour effectuer l'indexation d'images. Cependant il est également possible de les utiliser pour effectuer de la localisation d'objets, c'est-à-dire de détecter et positionner un objet qui ne recouvre que partiellement l'image. Chaque descripteur local est extrait dans une sous-partie de l'image délimitée par une fenêtre, notée F, qui est en général de taille constante. La localisation s'effectue en utilisant F comme une fenêtre glissante, c'est-à-dire que F va successivement tester toutes les positions de l'image globale. A chaque position de F, un descripteur sera extrait et permettra une classification.

Dans sa version la plus simple, un descripteur permettant de représenter le contenu de la fenêtre F peut être la liste des intensités des pixels qu'elle contient. Cependant ce choix n'est pas le plus pertinent. En effet, la description de l'imagette contenue dans F ne sera pas *robuste*, c'est-à-dire que le moindre changement sur l'imagette provoquera un changement sur le descripteur associé. Or les robustesses en traitement du signal et en traitement de l'image sont très importantes. Par exemple, nous souhaitons que le descripteur soit robuste aux rotations afin qu'il soit le même lorsqu'il est associé à une image

droite ou retournée. Les descripteurs doivent donc être choisis avec soin et leurs propriétés doivent être étudiées. Les représentations de l'imagette contenue dans F induisent différents types de robustesse, plus ou moins utiles en fonction du problème, et permettent d'ignorer des déformations géométriques et photométriques, comme les changements d'échelle, les variations de luminosité etc... (voir la figure 2.1)

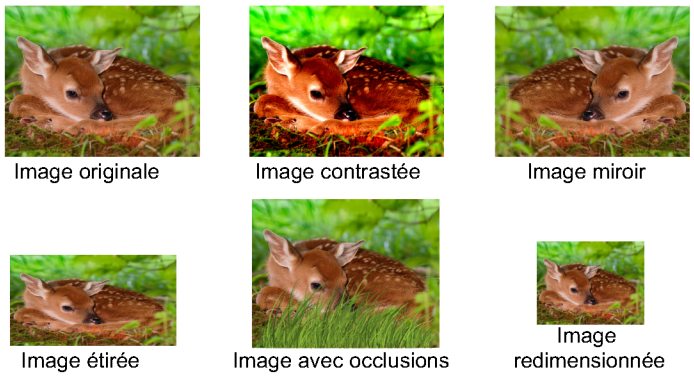


FIGURE 2.1 – Exemples d'images après différentes déformations géométriques et photométriques.

Dans ce chapitre nous introduirons trois grandes familles de descripteurs : les descripteurs photométriques, les descripteurs texturelles et les descripteurs par gradients. Dans le cas des descripteurs de textures et gradients nous ne considérerons pas l'ensemble des canaux couleur de l'image mais une image en niveaux de gris. Afin d'obtenir une image en niveaux de gris, plusieurs solutions sont possibles, mais nous utiliserons la luminance.

2.2 Descripteurs photométriques

Les descripteurs de photométrie, appelés descripteurs de couleurs, permettent de prendre en compte l'information colorimétrique de l'imagette contenue dans F. Dans cette section nous allons détailler le fonctionnement de deux descripteurs de couleur. Le premier construit un histogramme d'intensité à partir de chacun des canaux couleur. Le second se définit par un moment modélisant la distribution des intensités dans F. Chacun de ces descripteurs est très dépendant de l'espace couleur utilisé, dans notre cas l'espace couleur RGB. Dans une première section, nous montrons qu'il existe de nombreuses façons de normaliser cet espace couleur afin de gagner en robustesse.

2.2.1 Les propriétés des espaces couleurs

Dans la littérature de nombreux espaces couleurs existent et possèdent des propriétés différentes. Nous comparerons dans la section 6.3.1.3 plusieurs espaces couleurs connus comme les espaces HSV [57], CIE-LAB [83], etc... Dans cette section, nous considérons l'utilisation de l'espace couleur RGB et proposons différentes transformations afin d'obtenir des caractéristiques photométriques robustes.

Les auteurs de [43] ont montré qu'il est possible d'obtenir un espace invariant au changement de luminosité en normalisant les différentes composantes entres elles. Si nous notons $R,\ G,\ B$ les intensités respectives dans les canaux rouge, vert et bleu d'une image de base, l'équation 2.1 donne la transformation à effectuer. Cette transformation est particulièrement utile pour la détection d'objets se trouvant dans l'ombre de bâtiments, arbres, etc...

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} \frac{R}{R+G+B} \\ \frac{G}{R+G+B} \\ \frac{B}{R+G+B} \end{pmatrix} \tag{2.1}$$

Une seconde transformation, proposée par Van de Sande *et al.* [113], s'effectue sur chaque composante couleur indépendamment des autres. Elle permet d'être robuste aux changements de luminosité ainsi qu'au décalage par un offset constant. Elle consiste à normaliser les canaux en fonction de leurs moyennes et de leurs variances dans F, comme défini dans l'équation 2.2.

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} \frac{R-\mu_R}{\sigma_R} \\ \frac{G-\mu_G}{\sigma_G} \\ \frac{B-\mu_B}{\sigma_B} \end{pmatrix} \tag{2.2}$$

avec μ_x et σ_x la moyenne et la variance du canal x.

Une fois l'espace couleur défini, et normalisé si nécessaire, nous allons y extraire des caractéristiques. Dans la section suivante nous allons présenter le descripteur par histogrammes.

2.2.2 Descripteurs par histogrammes

Le descripteur par histogrammes 1D consiste à construire de manière décorrélée un histogramme pour chaque canal couleur dans la fenêtre F. Aussi pour décrire la couleur dans une fenêtre F, chaque histogramme 1D comptabilise les intensités pixels d'une composante couleur. L'ensemble des histogrammes est ensuite concaténé pour former un seul descripteur [43]. Ce descripteur ne considère pas les relations de voisinages entre les pixels et donc

l'information relative à la spatialité des pixels est perdue, on parle alors d'un descripteur avec un haut taux de certitudes.

Lors de problèmes de localisation, la perte totale de l'information spatiale est un défaut majeur. Pour contrecarrer cela, il est possible de diviser F en sous-régions qui sont appelées cellules. Un histogramme est alors calculé et normalisé sur chacune des cellules. L'ensemble des histogrammes est ensuite concaténé pour former le descripteur de couleurs final. Ainsi, pour conserver un minimum de spatialisation, de multiples histogrammes sont construits sur des parties de F et sont ensuite concaténés. Cependant le descripteur ainsi formé est de grande dimension et son utilisation est donc coûteuse. Dans la section suivante, nous introduisons un descripteur de couleurs possédant une taille plus compacte et donc étant plus exploitable pour la classification.

2.2.3 Descripteurs de moments

L'idée des descripteurs par moments est de considérer les pixels comme des triplets d'intensités (rouge, vert et bleu) formant une distribution dépendant de la position [75]. La distribution se caractérise par un moment dépendant de la position des pixels et par l'intensité des pixels en chacune des positions de la fenêtre F. Le moment, noté $M_{mq}^{a_1 a_2 a_3}$ (voir équation 2.3) est défini avec un ordre spatial, valant $m+q$, et un degré valant $a_1 + a_2 + a_3$ avec a_1, a_2, a_3 les degrés considérés pour les intensités couleurs rouges, vertes et bleues. Dans le travail de Mindru *et al.* [75], le descripteur de couleurs par moment est défini comme étant l'extraction des 27 moments suivants : $M_{mq}^{001}, M_{mq}^{010}, M_{mq}^{100}, M_{mq}^{002}, M_{mq}^{020}, M_{mq}^{200}, M_{mq}^{011}, M_{mq}^{101}, M_{mq}^{110}$ avec $\{m,q\} = \{\{0,0\},\{1,0\},\{0,1\}\}$. De plus, ils proposent de combiner linéairement les différents moments entre eux, formant 21 nouveaux descripteurs. Ces descripteurs ont des robustesses telles que l'invariance de couleurs aux changements de luminosité constants ou non dans les différents canaux.

$$M_{pq}^{a_1 a_2 a_3} = \iint_F x^m y^q I_{x,y,r}^{a_1} I_{x,y,g}^{a_2} I_{x,y,b}^{a_3} dxdy \tag{2.3}$$

2.2.4 Bilan sur les descripteurs photométriques

Pour les deux approches décrites précédemment, les intensités des pixels sont directement utilisées pour décrire l'imagette. Pour cela, la spatialisation n'est pas entièrement perdue mais aucune notion de voisinage n'est considérée. Ces descripteurs sont très performants dans le cas où la couleur est très discriminante ou en complément d'un second type de descripteur.

Dans notre cas, les tombes ne possèdent pas forcément de couleur significative permettant de les distinguer d'un autre type d'objet. Comme le montre la

figure 2.2, les tombes peuvent avoir une couleur très proche de celle d'autres zones de l'image globale : sol, potager, jardin... Il est alors intéressant de ne plus considérer seulement la couleur mais également les relations de voisinage entre les différents pixels de l'imagette. Pour cela, nous allons dans la prochaine sous-section introduire la notion de texture.

FIGURE 2.2 – Mise en avant de cas où la couleur est peu discriminante.

2.3 Descripteurs de textures

Actuellement il n'existe pas de définition générique de la texture. Cependant, la définition formelle provenant de la littérature [104, 109] tend à décrire la texture comme la représentation spatiale de motifs homogènes. Une texture se définit donc par une densité de répétitions de motifs, de formes et d'orientations. La figure 2.3 contient deux exemples de textures binaires (soit le pixel est noir, soit il est blanc), où le nombre de pixels noirs et blancs est constant. Sur

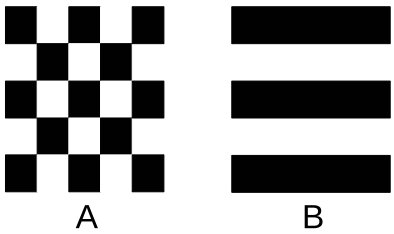


FIGURE 2.3 – Les deux figures contiennent deux textures avec la même quantité de pixels noirs et de pixels blancs.

l'image en damier 2.3.A, le motif qui se répète est de forme carrée et d'orientation fixe. Sur la seconde image 2.3.B, les motifs sont les lignes qui ont une orientation horizontale. Dans chacun des deux exemples synthétisés, l'homogénéité de la texture est "totale" et les frontières sont très clairement définies. Cependant, dans un cas de classification réelle de textures, comme sur la figure 2.4,

les motifs sont plus compliqués à déterminer et possèdent des frontières floues. Sachant cette contrainte nous allons chercher à décrire la texture à l'aide d'approches statistiques sans définir clairement le motif de la texture.

FIGURE 2.4 – Exemples de textures réelles provenant de la base de données *Describable Textures Dataset (DTD)*[17].

2.3.1 Les descripteurs par co-occurrence

Afin de décrire la texture, les descripteurs par calcul de co-occurrence supposent que l'on peut décrire la répartition d'intensité des pixels dans la zone comme un phénomène stochastique. En 1973, Haralick *et al.* [47] proposent d'extraire les propriétés statistiques entre les valeurs et les positions relatives des différents pixels. Dans un premier temps, une matrice de co-occurrence des pixels $C^{(\theta,d)}$, paramétrée par un angle θ et une distance d entre les différents pixels, est construite. Dans la matrice de co-occurrence sont comptabilisés, pour tous les pixels d'intensité i, le nombre de pixels d'intensité j se trouvant à la distance d dans la direction de θ. Par exemple, si nous prenons $\theta = 0$ et $d = 1$ nous considérons les pixels d'intensités j à droite d'autres intensités i. Le coefficient noté $C_{ij}^{0,1}$ représentera alors le nombre de fois où un pixel d'intensité j est à droite d'un pixel d'intensité i. Enfin, la matrice de co-occurrence $C^{(\theta,d)}$ est normalisée par le nombre de pixels dans la fenêtre F.

Dans un second temps, afin d'extraire les caractéristiques de textures nous allons calculer des propriétés statistiques sur la matrice $C^{(\theta,d)}$. Le tableau 2.1 décrit quelques propriétés statistiques fréquemment utilisées.

2.3.2 Les descripteurs par motifs binaires locaux

En 1995, un nouveau descripteur de textures a été proposé : les motifs binaires locaux (local binary patterns LBP) [48]. Ce descripteur permet une modélisation spatiale de la texture F sans utiliser de matrices de co-occurence. Il consiste en l'indexation de chaque pixel de F en une valeur dépendant de son voisinage. Une fois l'indexation réalisée en chaque pixel, un histogramme des index est construit sur la fenêtre F.

Afin de calculer l'index d'un pixel, dit pixel d'intérêt, à la position (x, y) il

Nom	Définition
maximum$^{(\theta,d)}$	$\max_{ij}(C_{ij}^{(\theta,d)})$
homogénéité$^{(\theta,d)}$	$\sum_i \sum_j \frac{(C_{ij}^{(\theta,d)})^2}{1+\lvert i-j\rvert}$
$moment_k^{(\theta,d)}$	$\sum_i \sum_j (C_{ij}^{(\theta,d)}).(i-j)^k$

TABLE 2.1 – Exemple de caractéristiques texturelles calculées sur une matrice de co-occurrence $C^{(\theta,d)}$. Pour le descripteur par *moment*, l'ordre est donné par la valeur de k.

convient de définir une distance d'interaction avec son voisinage à une distance d. Nous notons $x_i^{(d)}$ et $y_i^{(d)}$ la position d'un voisin, à une distance d, du pixel d'intérêt, avec $i \in \{0..N^{(d)}\}$ et $N^{(d)}$ le nombre de voisins situés à une distance d du pixel d'intérêt. La valeur d'indexation du pixel (x, y), notée $ind_{x,y}^{(d)}$, est obtenue en comparant sa valeur d'intensité à celle de ses voisins aux positions $(x_i^{(d)}, y_i^{(d)})$. Si la valeur d'intensité du pixel d'intérêt est supérieure à celle de ses voisins alors nous incrémentons la valeur de $ind_{x,y}^{(d)}$ en fonction de la position du voisin, sinon la valeur de $ind_{x,y}^{(d)}$ ne change pas, voir figure 2.5. Plus formellement l'indexation se calcule suivant l'équation :

$$ind_{x,y}^{(d)} = \sum_{i=0}^{N^{(d)}} \begin{cases} 2^i & si\ I_{x,y} \geq I_{x_i,y_i} \\ 0 & si\ I_{x,y} < I_{x_i,y_i} \end{cases} \tag{2.4}$$

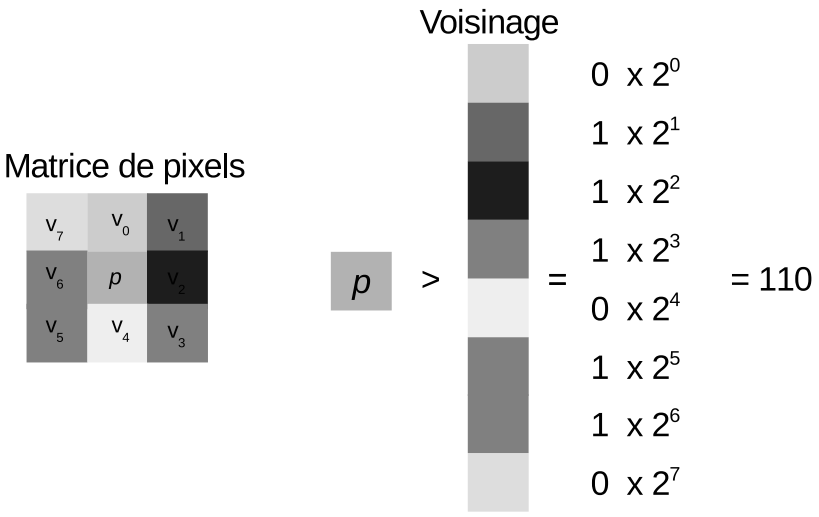


FIGURE 2.5 – Schéma donnant le code d'indexation LBP d'un pixel d'intérêt noté p entouré de 8 voisins.

Un histogramme est ensuite construit puis normalisé sur l'ensemble des valeurs d'indexation de tous les pixels dans F pour une distance d donnée. Afin d'augmenter la robustesse de ce descripteur et de décrire plus finement

le contenu d'une fenêtre F, il est possible de découper F en un ensemble de cellules. Un histogramme LBP est alors construit sur chaque cellule. Puis, l'ensemble des histogrammes est ensuite concaténé pour former le descripteur LBP final.

Le descripteur LBP peut être étendu lui permettant notamment d'avoir une invariance aux rotations [32]. Pour cela, lors de l'assignement de la valeur d'index à un pixel, nous faisons tourner le voisinage du pixel afin d'obtenir la valeur d'index la plus élevée possible. Ce descripteur est, encore aujourd'hui, très utilisé pour la description de textures [62] et particulièrement pour des applications de détection, comme la détection de visages [77].

2.3.3 Bilan sur les descripteurs textures

Les descripteurs par texture permettent une bonne description des relations entre voisinage au sein d'une imagette contenue dans F. Cependant les coordonnées des pixels sont perdues au profil de leurs positions relatives. Or, dans notre cas d'application la texture des tombes n'est pas assez prononcée (voire inexistante) et peut être assimilée à une structure de pierres uniformes qui ressemble à celle des trottoirs, des toits ou des murs larges. Sur la figure 2.6 on voit qu'il n'est pas possible de reconnaître les tombes uniquement en utilisant les relations de voisinage. Cependant, les contours peuvent être un bon indi-

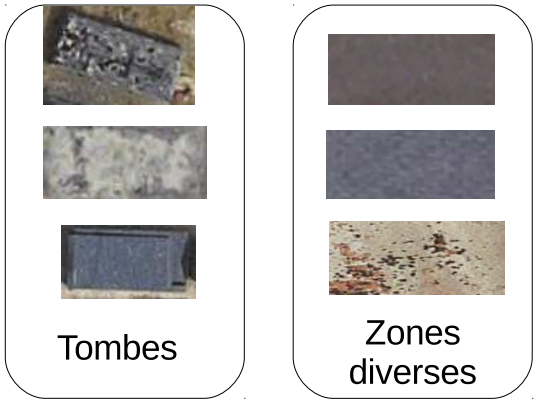


FIGURE 2.6 – Mise en avant des cas où la texture est peu discriminante.

cateur de la présence de tombes. L'ensemble des contours met en relief les fortes discontinuités de l'imagette et permet une analyse de *formes*. Ainsi, la présence de formes dans l'objet tel que les bords, croix, pots de fleurs, etc... seront représentés. Dans la sous-section suivante, nous décrirons le fonctionnement d'un descripteur de formes basé sur l'analyse du gradient.

2.4 Descripteurs de formes par analyse du gradient

2.4.1 Généralités sur les descripteurs de gradients

Actuellement de nombreuses compétitions académiques ont pour objectif de localiser et reconnaître automatiquement un ou plusieurs objets dans des images [35, 28]. La description des contours de l'objet va alors avoir une importance primordiale pour cette détection. La détection de contours est réalisée via l'étude des discontinuités d'intensité des pixels par rapport à leurs voisinage. Le plus souvent cette étape est faite en analysant une image résiduelle. Une fois les contours extraits, les descripteurs dits de "formes" permettent une représentation de l'arrangement spatial et des discontinuités d'intensité.

Ces descripteurs analysent le plus souvent deux images dérivées : une image horizontale et une autre verticale. Nous noterons $D^{(x)}_{x,y}$ la dérivée horizontale de $I_{x,y}$ et $D^{(y)}_{x,y}$ la dérivée verticale :

$$D^{(x)} = F \star I \;\; et \;\; D^{(y)} = F^T \star I \tag{2.5}$$

avec $\star$ l'opérateur de convolutions et F le filtre de contour utilisé, par exemple $F = [1, -1]$.

À partir des images dérivées il possible de calculer le module et l'orientation de chaque pixel, comme nous le voyons sur la figure 2.7. Nous notons $G_{x,y}$ et $\theta_{x,y}$ le module et l'orientation du pixel à la position (x, y), de façon à ce que $\theta_{x,y} \in]-\frac{\pi}{2}, \frac{\pi}{2}[$:

$$G_{x,y} = \sqrt{(D^{(x)}_{x,y})^2 + (D^{(y)}_{x,y})^2} \;\; et \;\; \theta_{x,y} = atan\left(\frac{D^{(x)}_{x,y}}{D^{(y)}_{x,y}}\right) \tag{2.6}$$

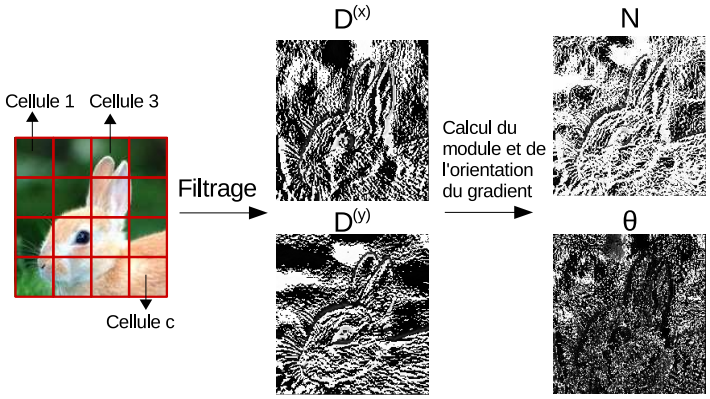


FIGURE 2.7 – Illustration du filtrage vertical et du filtrage horizontal puis calcul de la norme du gradient et de son orientation.

À partir des matrices de normes et d'orientations, de nombreuses méthodes permettent l'extraction de descripteurs [9, 10, 25]. Dans cette thèse, nous nous sommes particulièrement intéressés au descripteur HOG [26] qui présentaient de bonnes performances lors des stages qui ont précédé la thèse (comme nous

le verrons dans la section 3.4). La section suivante introduit et explique le fonctionnement d'un descripteur HOG.

2.4.2 Histogramme de Gradient Orienté

Le descripteur d'Histogramme de Gradient Orienté (HOG) permet de caractériser une forme locale en comptabilisant les différentes orientations des pixels dans une fenêtre F [26]. On parle de descripteur dense car, lors de sa construction, l'ensemble des pixels dans F va "voter" pour une direction et donc le descripteur HOG est calculé à partir de la totalité des pixels de F.

Chaque pixel à la position (x, y) vote pour une direction de l'histogramme. L'orientation donnée est cependant quantifiée en un nombre fini de directions, noté N_θ. Chaque angle $\theta_{x,y} \in]-\frac{\pi}{2}, \frac{\pi}{2}[$ est quantifié vers une des N_θ orientations. Nous notons $c_{x,y} \in \{0, N_\theta - 1\}$ le choix d'orientation du pixel à la position (x, y), comme l'indique l'équation suivante :

$$c_{x,y} = floor\left(\left(\theta_{x,y} + \frac{\pi}{2}\right)\frac{N_\theta}{\pi}\right) \tag{2.7}$$

avec $floor(x)$ la fonction retournant l'arrondi inférieur du réel x.

Le nombre de directions N_θ détermine la finesse de la description angulaire du descripteur HOG. En effet, plus N_θ est grand, moins l'information angulaire de chaque pixel est quantifiée et plus la précision augmente. Améliorer la précision de quantification provoque une augmentation de la taille du vecteur HOG. Si N_θ est trop grand alors l'information n'est quasiment plus quantifiée et le vecteur HOG deviendra donc "sparse", c'est-à-dire contiendra beaucoup de valeurs nulles. Dans la littérature, il est courant d'utiliser $N_\theta = 9$ soit une quantification angulaire de 40°.

Une fois la direction $c_{x,y}$ calculée, la composante correspondante dans le vecteur HOG est incrémentée en fonction de l'intensité locale du pixel, c'est à dire la norme du gradient $G_{x,y}$. Lorsque tous les pixels ont voté pour une orientation, il est nécessaire de normaliser le vecteur HOG.

Cependant la quantification en histogrammes des votes provoque une perte de l'information spatiale dans F. Afin de conserver cette notion il convient de diviser la fenêtre F en cellules de tailles inférieures et de calculer un histogramme HOG à l'intérieur de chacune des cellules. A l'intérieur de F, nous avons donc une *cellule glissante* qui extrait en chaque position un descripteur HOG. Dans [26] il est proposé de ne pas normaliser les descripteurs HOG des cellules mais de les regrouper en *blocs*. Chaque bloc est un agrégat de cellules adjacentes qui concatène les descripteurs HOG des cellules en un vecteur noté v. Afin de normaliser v, plusieurs solutions sont possibles et mènent à des per-

formances différentes en fonction du type d'images à traiter. Les trois plus utilisées, donnant des performances proches pour la détection de personnes, sont :

— la normalisation L1, $v = v/(||v||_1 + \epsilon)$,
— la normalisation L2, $v = v/(||v||_2 + \epsilon)$,
— la normalisation L2-hys [67]. Celle-ci consiste à normaliser le vecteur v suivant une norme L2, puis à tronquer ses valeurs lorsqu'elles excédent un seuil (par exemple 0.2) et enfin à re-normaliser le vecteur v. Cette normalisation permet notamment de lisser des zones où le gradient dans une direction est trop important.

L'ensemble des histogrammes v issus des blocs est ensuite concaténé formant le vecteur HOG final.

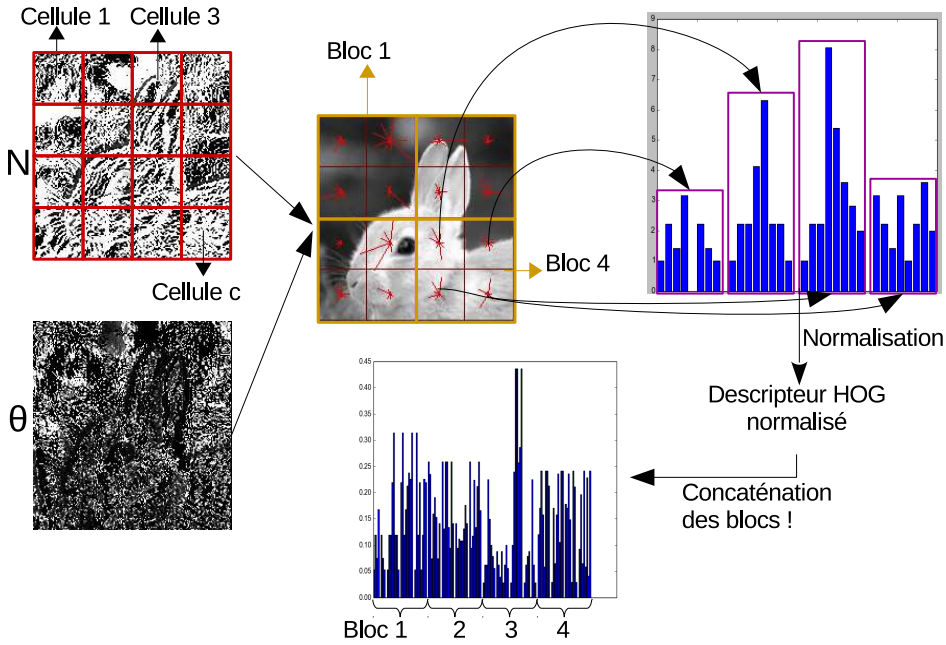


FIGURE 2.8 – Illustration de l'extraction du descripteur HOG. Dans cet exemple, les HOG sont calculés sur une grille constituée de 4×4 cellules. Une fois calculés, ils sont concaténés et normalisés selon un bloc regroupant les cellules par quatre donnant des blocs de deux par deux cellules.

Cependant le HOG calculé est très dépendant de la taille des cellules et donc de celle des blocs. En 2006, Zhu *et al.* [123] ont proposé de construire le descripteur HOG à partir de différentes tailles de bloc pour calculer le vecteur HOG à différentes résolutions ou échelles. La procédure pour calculer ce HOG multi-résolutions peut se diviser en trois étapes :

1. le nombre de cellules contenues dans chaque bloc est fixé. Dans la littérature, chaque bloc est le plus souvent partitionné en 2×2 cellules.
2. À l'aide d'une cellule glissante les HOG normalisés sont extraits et concaténés dans tous les blocs dont la taille est définie dans l'étape précédente.

3. La taille des blocs varie horizontalement puis verticalement afin de tester toutes les combinaisons de tailles possibles. Puis l'étape 2 est relancée.

Le descriptif du HOG est donc effectué à différentes résolutions augmentant considérablement sa représentabilité mais également sa taille. Si nous considérons des blocs constitués de 2×2 cellules, de taille allant de $L_x^{min} \times L_y^{min}$ à $L_x^{max} \times L_y^{max}$ avec un pas horizontal dx, et un pas vertical dy ; ainsi qu'une cellule glissante se déplaçant avec un chevauchement de 50%, la formule suivante donne la taille du vecteur, notée $|\mathbf{h}|$ avec $\mathbf{h}$ le vecteur HOG final :

$$|\mathbf{h}| = 4N_\theta \sum_{x=L_x^{min}}^{L_x^{max}} \sum_{y=y_x^{min}}^{L_y^{max}} floor\left(4\frac{t_F^2}{x \times y} dx\ dy\right) \tag{2.8}$$

avec t_F la largeur de la fenêtre F qui est ici considérée comme carrée.

2.4.3 Optimisation des calculs

Un système effectuant une détection d'objets a besoin de calculer le descripteur de très nombreuses fois. Il est alors primordial que le coût d'extraction devienne négligeable afin d'obtenir une application en temps réel. En 2001, Viola et Jones [116] démocratisent le concept d'image intégrale [23] afin de réduire considérablement le temps de calculs de leurs caractéristiques. Ce même concept d'images intégrales est également utilisé pour approximer des caractéristiques comme les descripteurs SURF ou Shape Context [3]. Détaillons comment l'image intégrale peut-être utilisée pour réduire le coût en calculs d'un descripteur HOG.

Nous définissons l'image intégrale, notée I', comme la représentation d'une image I où chaque point contient la somme des pixels situés au-dessus et à gauche de lui-même, comme l'indique l'équation suivante :

$$I'_{x,y} = \sum_{x' \leq x, y' \leq y} I_{x',y'} \tag{2.9}$$

A l'aide de cette représentation il devient très peu coûteux de calculer des sommes de pixels dans une zone rectangulaire donnée dans I. Par exemple, si nous considérons deux points de coordonnées (x_1, y_1) et (x_2, y_2), calculer la somme de toutes les intensités contenues dans le rectangle défini par ces points revient à effectuer trois additions, voir équation 2.10.

$$\sum_{\substack{x_1 < x \leq x_2 \\ y_1 < y \leq y_1}} I_{x,y} = I'_{x_1,y_1} + I'_{x_2,y_2} - I'_{x_2,y_1} - I'_{x_1,y_2} \tag{2.10}$$

Dans le cas du descripteur HOG, on constate que l'on calcule des sommes de normes de gradient dans des blocs en fonction d'une orientation donnée. Afin

de bénéficier de l'image intégrale, nous calculons pour chaque orientation notée $c \in \{0, N_\theta - 1\}$, une image contenant la norme des gradients des pixels. Ainsi, nous notons $G^{(c)}$ l'image contenant les normes des gradients dont l'orientation est c, tel que :

$$G^{(c)}_{x,y} = \begin{cases} G_{x,y} & \text{si } c_{x,y} = c \\ 0 & \text{sinon} \end{cases}$$

Nous notons $G^{(c)'}$ l'image intégrale de $G^{(c)}$. Le calcul d'un histogramme HOG dans une cellule donnée, bornée par les points de coordonnées (x_1, y_1) et (x_2, y_2) peut donc se simplifier. Chaque composante c de l'histogramme se calcule suivant l'équation : $G^{(c)'}_{x_1,y_1} + {G^{(c)'}}_{x_2,y_2} - {G^{(c)'}}_{x_2,y_1} - {G^{(c)'}}_{x_1,y_2}$. Aussi, une fois l'image intégrale calculée, le nombre d'opérations nécessaires pour calculer l'histogramme d'une cellule est $3 \times N$ additions.

2.4.4 Bilan sur les descripteurs de formes

Les descripteurs de formes décrits précédemment semblent pertinents pour la détection d'objets urbains [87, 88]. En effet, comme nous l'avons montré, le HOG décrit l'objet à plusieurs résolutions. A partir d'une basse résolution il met en avant les contours des tombes. Par contre, avec une haute résolution il décrit la présence d'objets sur la plaque tombale et la position de ces objets. De plus, ce descripteur peut se combiner avec d'autres descripteurs tel que le LBP pour améliorer les performances [118]. Dans le cas de la détection de tombes, cela n'est pas nécessaire car la texture est difficilement caractérisable mais cette démarche reste intéressante pour d'autres objets urbains.

Dans le cas de l'analyse d'images, certaines parties de l'image peuvent être présentes de nombreuses fois avec de légères variations. Ces parties d'images peuvent être caractérisées par des formes locales comme des contours spécifiques ou avoir un plus haut niveau d'abstraction comme des fleurs, pots, croix, plaques verticales, etc... Il peut être intéressant de chercher cet ensemble de caractéristiques répétitif afin de créer un descripteur listant leurs présences. Ces parties d'images forment ce que l'on nomme des "mots visuels". Dans la partie suivante, nous décrirons quelques méthodes basées sur la détection par des sacs de mots visuels.

2.5 Sac de mots visuels

2.5.1 Introduction et définitions

Afin de caractériser la fenêtre F deux choix sont possibles. D'une part nous pouvons extraire un descripteur décrivant la totalité de F qui est ensuite uti-

lisé pour la classification. D'autre part, nous pouvons utiliser une sous-fenêtre glissante permettant d'extraire un ensemble de descripteurs, noté $\mathcal{V}$, sur des parties de F. Dans le second cas, nous pouvons quantifier chaque descripteur de $\mathcal{V}$, en un ou plusieurs mots visuels [70]. Un mot visuel est la représentation d'un groupe (*cluster*) de motifs particuliers qui sont appris de façon non supervisée sur la base d'apprentissage. L'ensemble des mots visuels appris forment le dictionnaire de mots visuels, c'est-à-dire un ensemble de groupes permettant d'organiser une collection de motifs et formes locales présents sur les images de la base d'apprentissage. Afin de caractériser F, il est alors possible d'utiliser son apparence c'est à dire la fréquence d'apparition des mots visuels. Pour cela, chaque mot visuel est associé à un entier positif représentant un indice dans un histogramme h. La fenêtre F est donc représentée par un histogramme comptabilisant les indices associés aux mots visuels quantifiés pour chaque vecteur présent dans $\mathcal{V}$.

Les avantages de l'approche par sac de mots sont multiples. Le plus souvent la taille d'un seul vecteur de caractéristiques extrait sur la globalité de la fenêtre F est plus grande que la taille du dictionnaire. Aussi, l'utilisation de l'histogramme h, décrivant F, permet une réduction de la dimension du vecteur représentatif et donc une diminution considérable du coût mémoire et facilite la recherche et l'indexation de ces images dans des bases de données [103, 38]. De plus, cette approche permet une quantification en motifs connus de la base et amène donc une robustesse pour regrouper des objets indiquant la même information bien qu'ils soient légèrement différents les uns des autres. Chaque fenêtre F est finalement décrite par la présence et l'absence des mots visuels contenus dans le dictionnaire.

La première étape est la construction d'un dictionnaire à l'aide de méthodes d'apprentissage non-supervisée. Cette étape permet à partir d'un ensemble de descripteurs, extraits sur la totalité de la base d'apprentissage, d'obtenir les K meilleurs représentants de ces vecteurs. Les représentants de la base vont ensuite être utilisés pour la quantification en mots visuels et forment donc le dictionnaire. Afin de trouver ces représentants, de nombreux algorithmes existent et sont utilisés avec des coûts et représentations différents [80, 76]. Un des algorithmes les plus connus est celui des K-moyens (*Kmeans*) qui divise les données en K partitions [78]. Le centre de chaque partition est appelé centroïde et est noté μ_i avec $\mathbf{i} \in \{1..K\}$. L'objectif est de trouver l'ensemble des centroïdes formant l'ensemble $\mathcal{U}$ qui est utilisé comme dictionnaire de mots visuels.

Afin de décrire son fonctionnement, nous notons $\mathbf{x} \in S_i$ un descripteur appartenant à l'ensemble S_i. Cet ensemble S_i contient tous les descripteurs extraits dont le centroïde associé est le numéro i, c'est-à-dire les descripteurs dont la distance au centroïde μ_i, notée D, est minimale. A chaque itération l'algo-

rithme cherche à minimiser l'écart entre les centroïdes de $\mathcal{U}$ et ses observations associées comme décrit par l'équation 2.11.

$$\mathcal{U} = \underset{\mathcal{U}}{argmin} \sum_{i=1}^{K} \sum_{\mathbf{x} \in S_i} D(\mathbf{x}, \mu_i) \tag{2.11}$$

La seconde étape est la quantification de chaque descripteur en un histogramme de mots visuels. La figure 2.9 résume l'étape de quantification, où l'on extrait un ensemble de descripteurs dans la fenêtre glissante (rectangle rouge) qui sont ensuite quantifiés en un point de couleur. La fenêtre est finalement décrite par l'histogramme comptabilisant le nombre de points de chaque couleur. L'histogramme h est normalisé par une norme L1 afin d'obtenir la fréquence d'apparition des mots visuels dans F.

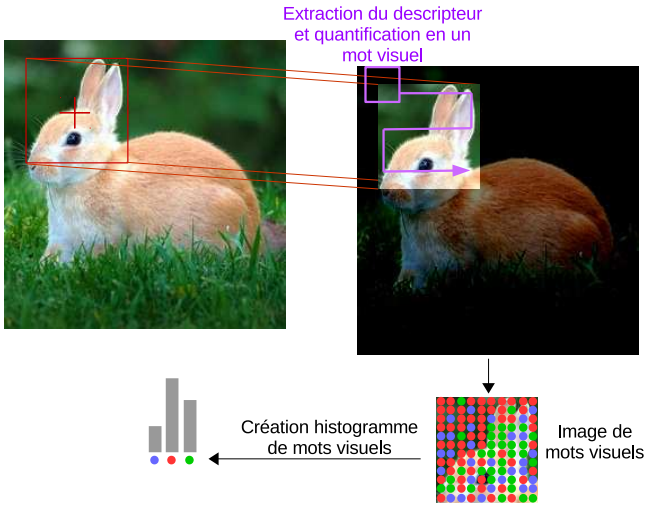


FIGURE 2.9 – Extraction de l'histogramme de mots visuels caractérisant la fenêtre rouge.

Notons que la quantification des mots visuels en histogrammes provoque une perte de l'information spatiale. Afin de conserver la géométrie de l'objet, l'utilisation d'une représentation pyramidale a été introduite par Lazebnik [99]. L'idée est de partitionner spatialement et récursivement l'image en régions de plus en plus petites jusqu'à une profondeur L. Un histogramme de mots visuels est ensuite construit à l'intérieur de chacune des sous régions. L'ensemble des histogrammes est concaténé et pondéré en fonction de la profondeur de sa région, notée $l \in \{0, L\}$, selon la loi suivante : $w_l = \frac{1}{2^{L-l}}$.

2.5.2 Vecteur de descripteurs agrégés localement

Une autre approche consiste non pas à créer un histogramme h mais à stocker les écarts entre les descripteurs et les centroïdes représentant les mots visuels. La méthode VLAD (*Vector of Locally Aggregated Descriptors*) consiste à

représenter l'imagette F par l'aggrégation des sommes des différences entre les descripteurs locaux et les centroïdes associés [56]. Nous rappelons que le dictionnaire est composé de centroïdes $\mathcal{U} = \{\mu_0..\mu_K\}$ avec μ_k le $k^{ième}$ mot visuel et K le nombre de mots visuels, et $\mu_k = NN(\mathbf{x}_i)$ une fonction retournant le mot visuel le plus proche d'un descripteur quelconque $\mathbf{x}_i$, tel que :

$$NN(\mathbf{x}_i) = \underset{\mu_k \in U}{arg\ min}\,(D(\mathbf{x}_i, \mu_k))$$

Nous posons $\mathbf{x}_i \in \mathcal{V}$ un descripteur extrait localement dans F. Nous calculons le vecteur descripteur VLAD, $\mathbf{v}_k$ pour chaque mot visuel suivant l'équation suivante :

$$\mathbf{v}_k = \sum_{\mathbf{x}_i \ tel\ que\ NN(\mathbf{x}_i)=\mu_k} \mathbf{x}_i - \mu_k$$

Pour chaque mot visuel présent dans le dictionnaire le vecteur $\mathbf{v}_k$ est calculé puis normalisé suivant une norme L2. L'ensemble des vecteurs $\mathbf{v}_k$ est ensuite concaténé pour former le descripteur VLAD final. Ce descripteur est particulièrement intéressant pour l'indexation d'images. En effet, même après une réduction de la dimension, à l'aide d'une Analyse en Composantes Principales (ACP), la représentativité de l'image est conservée [56, 7].

Actuellement, les approches de détections reposant sur les sacs de mots visuels sont nombreuses. L'extension utilisant des vecteurs agrégés offre de très bonnes performances [114] et a été reprise dans des architectures différentes telles que certaines architectures profondes [50] que nous détaillerons dans le chapitre 4. Une fois le descripteur extrait, il est nécessaire de le classifier. Pour cela, nous utilisons des algorithmes de classifications supervisées. Une fois le modèle appris, plusieurs stratégies permettant d'effectuer la détection dans l'image de tests sont possibles : l'approche par objet et celle par pixel. Dans la section suivante nous détaillerons ces différentes approches.

Chapitre 3

Classification pixel et classification objet

L'objectif de la classification supervisée est d'établir un ensemble de règles, formant un modèle, qui vont permettre de définir des frontières entre des objets de natures différentes. Pour cela, la première étape consiste à décrire les objets de manière fine. La seconde étape consiste à construire un modèle, à partir d'objets connus, c'est à dire d'objets dont nous connaissons à l'avance la nature. La nature d'un objet est appelée la classe, le label ou encore l'étiquette.

Dans le chapitre précédent nous avons expliqué différentes façons de représenter une imagette contenue dans F par le biais de caractéristiques. Pour cela, nous pouvons calculer des descripteurs permettant de représenter différentes informations comme la couleur, la texture et les contours. Nous pouvons également transformer les descripteurs en un histogramme de mots visuels ce qui permet de connaître la fréquence d'apparition de motifs dans l'imagette. Il est désormais nécessaire de classer chaque descripteur pour connaître son type d'objets contenu dans F. Dans ce chapitre, nous noterons $\mathbf{d}_i$ un vecteur de caractéristiques et t_i son étiquette associée, avec $i \in \{0, N-1\}$ et N le nombre de vecteurs dans la base d'apprentissage.

Pour effectuer la classification nous utilisons un classifieur effectuant un apprentissage supervisé. Nous allons dans une première partie expliquer le fonctionnement d'un classifieur très utilisé, à savoir le séparateur à vaste marge (SVM). Puis, nous décrirons l'utilisation du classifieur pour détecter des objets dans une image de tests. Pour cela, nous présenterons deux approches à savoir l'approche par pixel et celle par objet. Pour chacune de ces approches, nous détaillerons une méthode permettant une accélération des temps de détection afin d'obtenir des applications en temps réel. Enfin dans une dernière partie, nous présenterons les comparaisons effectuées entre l'approche par pixel et par objet et nous proposerons une contribution permettant d'améliorer l'efficacité de l'approche objet.

3.1 Les Séparateurs à Vaste Marge

Le séparateur à vaste marge (SVM) est un algorithme permettant de trouver la meilleure séparation entre deux ensembles de données. Pour cela, nous cherchons dans une dimension donnée le meilleur hyperplan séparateur séparant deux jeux de données. Afin de trouver une séparation binaire le SVM calcule un vecteur appelé poids qui est orthogonal au plan séparateur, noté $\mathbf{w}$, obtenu par la minimisation de l'expression suivante :

$$\min_{\mathbf{w}} \mathcal{R}(\mathbf{w}) + C\sum_{i=0}^{N}\mathcal{L}(\mathbf{w}, \mathbf{d}_i, t_i) \tag{3.1}$$

avec C un paramètre réel de contraintes défini par l'utilisateur, $\mathcal{L}$ la fonction de perte et $\mathcal{R}$ la fonction de régularisation. Le choix de la fonction de perte est important et dépend du problème. Elle permet de régler le degré d'ajustement des données en mesurant la différence entre la sortie prédite et la valeur attendue. Une des fonctions de régression la plus connue [18] est la fonction logistique qui se définit comme étant $\mathcal{L}(\mathbf{w}, \mathbf{d}, t) = log(1 + e^{-t\mathbf{w}^T\mathbf{d}})$. Dans *liblinear* [36], deux fonctions de régularisation existent, à savoir la régularisation L_1 définie dans 3.3, et la régularisation L_2 voir l'équation 3.2.

$$\mathcal{R}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{3.2}$$

$$\mathcal{R}(\mathbf{w}) = |\mathbf{w}|_\mathbf{1} \tag{3.3}$$

Afin de prédire l'appartenance d'un nouveau vecteur à une classe il suffit d'effectuer le produit scalaire entre sa caractéristique, notée $\mathbf{d}_{test}$, et le vecteur poids. Le score retourné, noté s, est calculé comme :

$$s = \mathbf{w^T}.\mathbf{d}_{test} \tag{3.4}$$

Ce score peut-être normalisé à l'aide d'une fonction sigmoïde dont le but est de représenter une probabilité entre 0 et 1 [41]. Dans le cas binaire, nous disposons de deux classes A et B, la classe retournée est celle avec la probabilité associée la plus grande. Ainsi si la probabilité est proche de 0, le vecteur est considéré comme appartenant à la classe A sinon il est considéré comme faisant partie de la classe B. Dans le cas multi-classes, plusieurs stratégies existent pour obtenir une probabilité pour chacune des classes. Par exemple, la stratégie *one-against-one* [120] consiste à créer plusieurs modèles afin d'effectuer toutes les combinaisons binaires de classes possibles.

Cependant, les problèmes réels sont très rarement linéairement séparables dans l'espace de définition de leurs descripteurs. Pour résoudre ces problèmes avec un hyperplan, la solution consiste à transformer l'espace des données dans

un nouvel espace où ils deviendront linéairement séparables. Le nouvel espace ainsi défini s'appelle l'espace de *redescription*. Effectuer la transformation vers l'espace de redescription est une tâche coûteuse qui n'est pas nécessaire lorsque l'on utilise le *kernel trick* . Afin d'illustrer cette technique de redescription et l'astuce associée, considérons l'exemple simple suivant :
Nous notons $\mathbf{x}$ et $\mathbf{y}$ deux vecteurs descripteurs dans $\mathbb{R}^2$. Afin de résoudre le problème de séparation, il est nécessaire d'effectuer une transformation vers l'espace de redescription ϕ tel que $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ à l'aide de la transformation suivante : $\phi(\mathbf{x}) = (\mathbf{x}_1^2 + \mathbf{x}_2^2 + \sqrt{2}\mathbf{x}_1\mathbf{x}_2)$. Calculons le produit scalaire dans l'espace de redescription :

$$\begin{aligned} \phi(\mathbf{x}).\phi(\mathbf{y}) &= (\mathbf{x}_1^2\mathbf{y}_1^2 + \mathbf{x}_2^2\mathbf{y}_2^2 + \sqrt{2}\mathbf{x}_1\mathbf{x}_2\sqrt{2}\mathbf{y}_1\mathbf{y}_2) \\ &= (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)^2 \\ &= (\mathbf{x}.\mathbf{y})^2 \end{aligned} \tag{3.5}$$

Nous constatons que le calcul du produit scalaire dans ϕ peut se faire sans effectuer la transformation. Le *kernel trick* consiste à utiliser une fonction noyau K tel que $K(d_i, d_j) = \phi(d_i).\phi(d_j)$. Dans l'exemple présenté dans l'équation 3.5, la fonction noyau est une fonction polynomiale de degré 2, définie par $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}.\mathbf{y})^2$. Une des fonctions noyau la plus utilisée est la fonction radiale $K(x, y) = e^{-\frac{(x-y)^2}{2.\sigma^2}}$.

La complexité des SVM est très importante, de l'ordre de $O(|\mathbf{d}|.N^2)$ avec $|\mathbf{d}|$ la taille du descripteur et N la taille de la base d'apprentissage [15]. Dans notre cas, afin de couvrir toute la diversité des objets à détecter nous considérons un grand nombre d'échantillons à traiter (N est très grand) rendant ce type de méthode inutilisable. Nous employons donc une approximation de SVM implémentée dans la bibliothèque *liblinear* que nous nommerons SVM linéaire [36]. Les SVM linéaires cherchent à résoudre le problème sans effectuer de transformations à l'aide de l'espace de redescription. Ils possèdent donc une complexité de l'ordre de $O(N)$. Ils sont utilisés pour traiter les grands volumes de données avec plusieurs millions de descripteurs et sont particulièrement efficaces pour les problèmes dit *sparses* [2].

3.2 Méthodologie de classification pixel

3.2.1 Fonctionnement de la classification pixel

La classification de type pixel consiste à attribuer à chaque pixel de l'image un label ou une probabilité d'appartenance à une classe. Pour cela, chaque pixel va être classifié en fonction de son voisinage. Lors de l'extraction de caractéristiques en un pixel dit d'intérêt, noté p, nous considérons une fenêtre F de taille constante centrée sur celui-ci. Plus la taille de F est grande, plus l'infor-

mation du voisinage sera prise en compte. Lors d'une classification pixel, il est alors classique d'utiliser la quantification en mots visuels qui permet à chaque pixel de la fenêtre F d'être rattaché à un ou plusieurs motifs connus [66, 2, 3]. Le vecteur caractéristique extrait, noté $\mathbf{x}$, est donc un histogramme des fréquences de mots visuels autour de ce pixel.

Suite à cette classification pixel, nous obtenons, pour chaque classe une carte de probabilités de même taille que l'image originale. Dans le cas d'un classifieur linéaire, tel que le SVM linéaire, le score de chaque pixel noté s vaut :

$$s = \frac{1}{||\mathbf{x}||} \sum_{i=0}^{K} \mathbf{x}_i \mathbf{w}_i + b \tag{3.6}$$

avec $\mathbf{w}$ le poids du classifieur linéaire, b le biais du classifieur et K le nombre de mots visuels dans le dictionnaire.

Un objet étant représenté par un groupe de pixels, il est donc nécessaire d'effectuer un post-traitement sur cette carte afin de différencier les objets entre-eux. Ce traitement est d'autant plus important que, dans notre cas, les objets urbains peuvent être collés et que l'on souhaite une détection individuelle de chacun d'entre eux. Plusieurs solutions ont été proposées, dont l'une des plus connues [84] consiste à appliquer un algorithme de *Mean Shift* [21]. L'algorithme du Mean shift est itératif et a pour objectif de faire converger chaque pixel de l'image vers une couleur représentant son maximum local. Il va ainsi partitionner de façon non supervisée l'image en classes. Chaque groupe de pixels est alors associé à la probabilité moyenne des pixels le composant.

3.2.2 Optimisation avec une image intégrale de la phase d'évaluation

Lors de la phase d'évaluation d'une image de taille $N \times M$, l'ensemble des pixels est évalué. Effectuer la classification pour chaque pixel est une tâche très coûteuse. Dans le cas d'un classifieur linéaire, il est nécessaire d'effectuer $N \times M$ le calcul du produits scalaires défini dans l'équation 3.6.

En 2009, les auteurs de [2] proposent de réduire la complexité calculatoire en utilisant le principe de l'image intégrale dans le cas d'un classifieur linéaire. L'objectif de leur approche est de classifier chaque pixel sans utiliser l'équation 3.6. Nous rappelons, qu'une image intégrale est une représentation d'images dans laquelle il est très rapide de calculer des sommes dans des zones rectangulaires, nous avons détaillé son fonctionnement dans la section 2.4.3. Dans un premier temps, il est nécessaire d'extraire le descripteur en tous points de l'image et de le quantifier en mots visuels. Dans un second temps, en utilisant un modèle linéaire, tel que le SVM linéaire, chaque mot visuel peut être associé à un score et donc chaque pixel est remplacé par son poids associé

dans le modèle d'apprentissage. Sur l'image constituée de poids, nous construisons l'image intégrale. Cependant, avant d'effectuer cette étape, il est nécessaire d'avoir une image contenant uniquement des poids positifs. Les valeurs du poids du classifieur sont modifiées avec la règle suivante : $\widetilde{\mathbf{w}} = \mathbf{w} - w_{min}$ avec w_{min} la valeur minimale du poids. En réinjectant les valeurs de poids modifiées dans l'équation 3.6 on trouve l'équation 3.7. Cette image de scores est représentée dans la figure 3.1.

$$s = \frac{1}{||\mathbf{x}||}\sum_{i=0}^{K}\mathbf{x}_i\widetilde{\mathbf{w}}_i + \frac{w_{min}}{||\mathbf{x}||}\sum_{i=0}^{K}\mathbf{x}_i + b \tag{3.7}$$

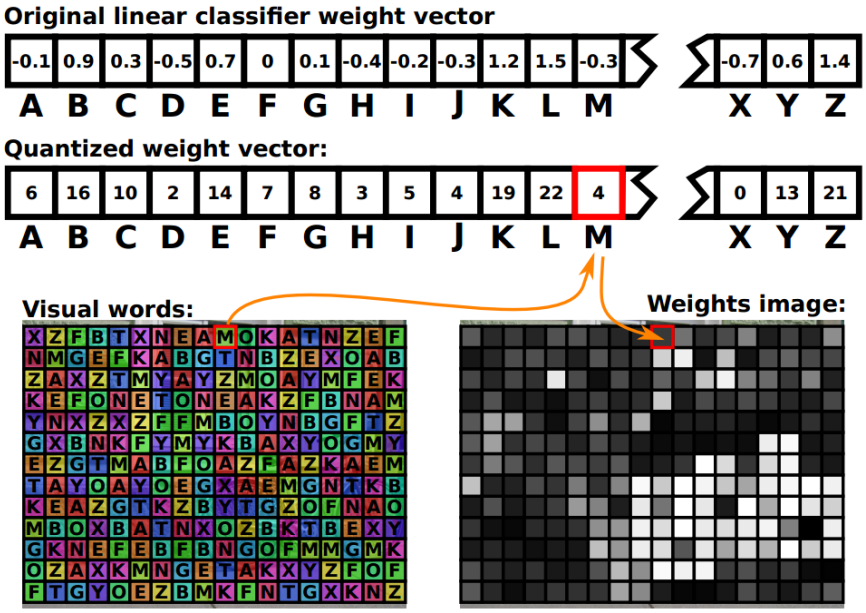


FIGURE 3.1 – Image (provenant de [3]) illustrant la construction de l'image de poids à partir de l'image de mots visuels.

La probabilité d'appartenance à une classe d'un pixel p est obtenue par la somme des probabilités d'appartenance à la classe des pixels dans son voisinage. Pour calculer cette somme il est alors possible d'utiliser l'image intégrale de l'image de scores. A partir de cette image intégrale le calcul du score final pour chaque pixel, en prenant en compte son voisinage, est de 3 opérations. Pour obtenir une probabilité normalisée, il est nécessaire de normaliser le descripteur. La normalisation L_1 consiste à diviser l'histogramme $\mathbf{x}$ par le nombre de mots visuels qui le compose. Or, dans le cas d'une quantification en mots visuels classique, tel que le K-means, chaque pixel est quantifié en un et un seul mot visuel. Il y a donc une occurrence de mots visuels par pixel dans l'histogramme et donc le calcul de s peut se simplifier, voir équation 3.8.

$$s = \frac{1}{t^2}\sum_{i=0}^{K}\mathbf{x}_i\widetilde{\mathbf{w}}_i + w_{min} + b \tag{3.8}$$

avec t la largeur de la fenêtre carrée centrée sur p.

Cette optimisation permet donc de s'affranchir du calcul de l'histogramme de mots visuels et donc du calcul du produit scalaire donné dans 3.6. Dans [2], les auteurs obtiennent un système de classification de 11fps contre 8fps pour la méthode [101] sur des images de dimensions 300×213 pixels.

3.3 Méthodologie de classification objet

3.3.1 Fonctionnement de la classification objet

Contrairement à l'approche pixel, l'approche objet cherche à attribuer une probabilité ou directement un label à une fenêtre F. Pour cela, nous créons des modèles permettant de définir les caractéristiques d'un objet dans sa totalité. Ce type d'approche est très largement utilisé, notamment pour la détection d'objets [116, 123]. Les modèles doivent décrire les objets de façon générique afin de les reconnaître dans des conditions différentes de celles présentes dans la base d'entraînement. Pour cela, la base de données doit contenir des exemples de l'objet avec des représentations différentes. Pour créer les différents modèles nous pouvons décomposer la méthode en trois points.

1. En utilisant la vérité terrain c'est-à-dire les images dont les positions des objets sont connues, les imagettes contenant les différents objets recherchés sont extraites. Dans notre cas de détection de tombes, nous obtenons donc deux listes d'imagettes : l'une contenant les tombes et la seconde des zones aléatoirement choisies ne contenant pas entièrement des tombes. Cependant, comme la taille des objets varie celle des imagettes n'est pas identique. Or, pour effectuer l'apprentissage, il est nécessaire que la dimension des descripteurs soit identique et représente la même information. Nous normalisons donc la taille de chaque imagette à une taille D constante. Le modèle appris est donc propre à une taille donnée comme l'illustre la figure 3.2.
2. Nous pouvons à partir des imagettes redimensionnées extraire une liste de descripteurs.
3. En connaissant le label des descripteurs nous pouvons utiliser des méthodes de classifications supervisées, comme les SVM, afin de trouver les meilleures frontières entre les différentes classes.

Durant la phase d'évaluation, le concept de fenêtre glissante est utilisé afin de localiser les objets sur l'image de tests. La fenêtre glissante, notée F, parcourt l'image et à chaque position extrait la probabilité que F appartienne aux différentes classes. Pour obtenir un descripteur homogène à celui utilisé lors de la phase d'apprentissage, F doit être de dimension D. Cependant les différents objets sur l'image de tests n'ont pas tous une dimension égale à D. Pour détecter

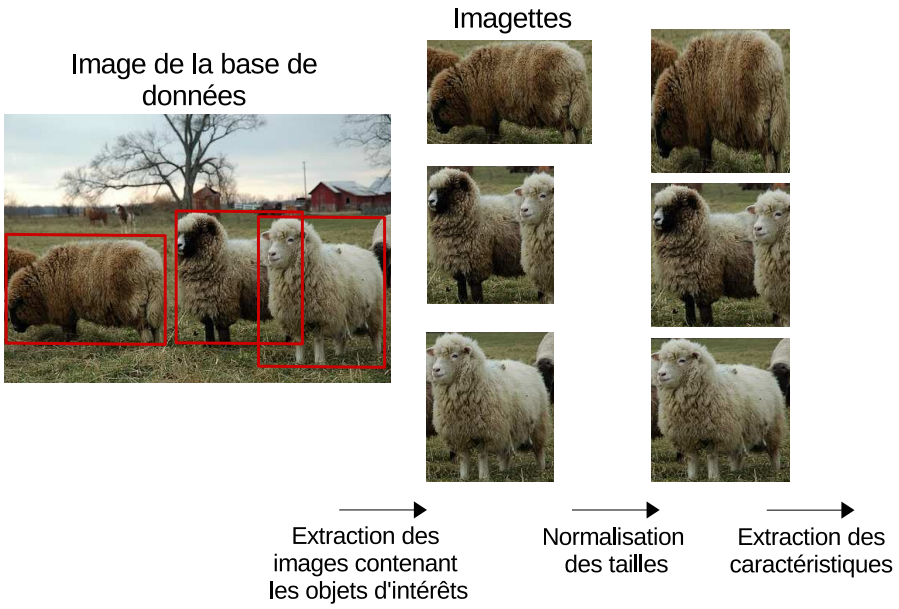


FIGURE 3.2 – Illustration de l'extraction et de la normalisation en taille des imagettes contenant des objets d'intérêt de type mouton.

les objets dont la taille n'est pas égale à D, il est possible de considérer des fenêtres glissantes de différentes tailles et de les redimensionner à chaque position. Cependant, ce processus est très coûteux. Pour éviter ce problème, nous redimensionnons directement l'image de tests. En effet, pour détecter des objets avec une taille supérieure à D il est nécessaire de réduire la taille de l'image de tests, et à l'inverse, pour détecter correctement des objets avec une taille inférieure à D il est requis d'agrandir l'image de tests.

Ainsi pour localiser tous les objets d'intérêt de tailles variables et inconnues dans une image de test, celle-ci doit être redimensionnée à différentes résolutions. La figure 3.3 illustre parfaitement ce problème, puisque nous constatons la présence d'un mouton et d'un agneau sur l'image provenant de [35]. Il s'agit alors de deux objets que l'on cherche à détecter comme étant de type mouton. Le mouton possède la même taille apparente que la fenêtre glissante F représentée en rouge, il est donc détecté sans changer la résolution de l'image. Cependant, pour que F contienne entièrement l'agneau il est nécessaire de déformer l'image de tests en multipliant par 1.5 sa largeur et 1.6 sa hauteur. A cette nouvelle résolution, l'agneau peut-être détecté. Une fois toutes les résolutions de l'image testées, l'ensemble des résultats est fusionné et ramené sur l'image unitaire où l'on peut voir les deux détections de mouton.

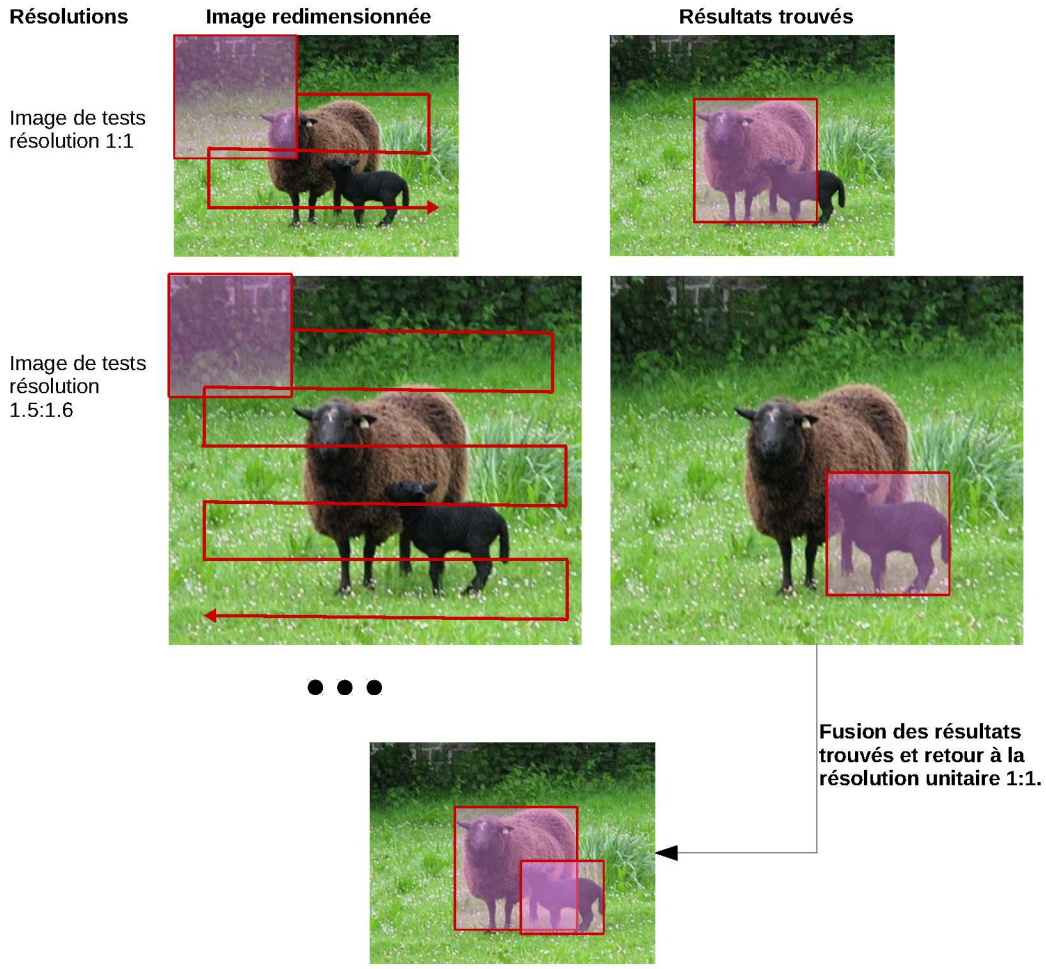


FIGURE 3.3 – Illustration de la nécessité de déformer l'image afin que tous les objets soient détectables dans une fenêtre de taille constante, ici représentée en rouge.

En testant toutes les positions à toutes les résolutions sur l'image de tests, le nombre de fenêtres devient très grand [94, 116]. Nous avons vu que l'extraction de caractéristiques pouvait être accélérée en utilisant le concept d'images intégrales, section 2.4.3. Cependant la partie classification reste coûteuse. Il est possible de l'accélérer en utilisant le concept de cascade, ce que nous allons voir par la suite.

3.3.2 Optimisation avec une cascade de classifieurs

Lors de la phase d'évaluation, effectuer le produit scalaire avec le vecteur poids du SVM reste une tâche coûteuse car d est de grande dimension. Pour réduire la complexité, il est possible d'utiliser une cascade de classifieurs [116, 123]. La concept de la cascade repose sur le fait que la plupart des imagettes testées lors de la phase de tests ne sont pas des objets d'intérêt. Par ailleurs, nous supposerons qu'il est possible avec un sous-ensemble de caractéristiques

de les éliminer. En se basant sur ce principe, il est nécessaire de construire une méthode éjectant le plus vite possible tous les faux positifs.

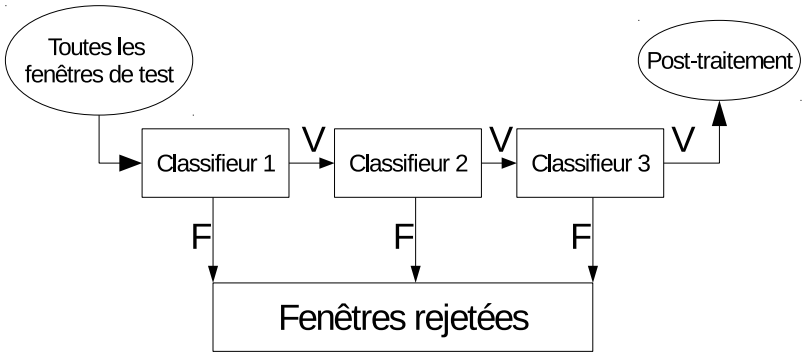


FIGURE 3.4 – Illustration montrant le fonctionnement d'une cascade. Lorsqu'une fenêtre est classée comme n'étant pas l'objet recherché elle est immédiatement rejetée. Dans le cas contraire elle passe au classifieur de rang supérieur.

La cascade est une succession de classifieurs, appelé *classifieurs faibles*, organisés avec un nombre croissant de caractéristiques. L'objectif de chaque classifieur est d'éliminer un pourcentage de faux positifs présents dans la base. Pour cela, le classifieur au rang $n+1$ considère comme base d'apprentissage toutes les données classées comme étant des objets d'intérêt par le classifieur de rang n. Pour chaque étage, l'utilisateur doit préciser le taux de fausses alarmes maximal, noté f_{max}, c'est à dire l'erreur permise par le classifieur ainsi que le nombre de détections minimales requises, noté d_{min}. Le protocole utilisé lors de l'apprentissage dépend du type de classifieur faible utilisé.

Dans la cascade [123] les auteurs utilisent des SVM pondérés à l'aide d'Adaboost [40]. Lors de l'apprentissage d'un étage, 5% des blocs du descripteur HOG sont aléatoirement utilisés pour l'entraînement d'un SVM. Si cet SVM suffit à atteindre le seuil d_{min} et que f_{max} est atteint, alors l'apprentissage de cet étage s'arrête et l'apprentissage de l'étage suivant commence. Sinon le SVM est rajouté à l'aide d'Adaboost à un nouveau SVM qui est calculé avec 5% des nouveaux blocs aléatoirement tirés. Cette étape de concaténation continue jusqu'à atteindre le seuil de détectabilité d_{min}.

3.4 Comparaison de la classification objet et pixel

3.4.1 Travaux préliminaires

Avant que cette thèse ne soit initiée, deux stages [22, 112] avaient tenté d'effectuer une détection de tombes sur des données similaires à celles que nous avons utilisées.

Le premier stage avait mis en place l'approche objet proposée par Viola et Jones [116]. Pour cela le descripteur pseudo-Haar était utilisé. Lors de l'apprentissage une cascade de 32 classifieurs effectue l'apprentissage de classifieurs 'faibles' de type Adaboost [39].

Lors du second stage une approche pixel a été utilisée en se basant sur les travaux de Aldavert *et al.* [3]. En utilisant le logiciel [1] développé par ces mêmes auteurs, il est facile de tester plusieurs types de classifications. Ainsi lors de ces travaux les descripteurs simplifiés HOG, SURF et *SHAPE CONTEXT* furent utilisés et comparés. Une quantification en mots visuels a également été effectuée à l'aide des algorithmes de *K-means hiérarchique* [80] et d'*Extreme Random Forest* (ERF) [76]. A l'issue de ces comparaisons, il a été montré que le descripteur HOG combiné avec une quantification ERF donnent les meilleurs résultats.

Afin d'obtenir un résultat exploitable dans le cas de la localisation d'objets urbains, les résultats de l'approche pixel sont segmentés en objets lors d'un post traitement. Ce post-traitement est effectué en utilisant un algorithme de Mean Shift à différentes résolutions et en fusionnant les différents résultats [3, 84].

	Approche Pixel	Approche Objet
Rappel	0.58	0.76
Précision	0.72	0.53
Fmesure	0.60	0.56

TABLE 3.1 – Résultats provenant de [16, 112] comparant une approche pixel avec une approche objet de [22] en utilisant une base de cimetières.

Le tableau 3.1 compare les résultats des deux approches décrites précedement. Aucun des deux cas n'est meilleur avec des F_{mesure} proches et les résultats ne permettent pas une application efficace. De plus, la méthode de classification objet réalisée durant le stage a été améliorée de nombreuses façons, notamment avec l'utilisation de descripteurs HOG remplaçant le descripteur pseudo-Haar. Dans la sous section suivante, nous avons comparé l'approche pixel à une approche objet plus récente et introduisons une première amélioration.

3.4.2 Comparaison des approches et améliorations

Pour se comparer correctement aux méthodes pixel décrites précédemment, nous proposons d'utiliser l'approche provenant de [123] utilisant des descripteurs HOG multi-résolutions avec une cascade de SVM. Par ailleurs, nous avons repris le protocole précédant en effectuant une *cross-validation* (décrit dans le chapitre 5) pour obtenir des résultats plus représentatifs. De plus, la base est

1. `http://www.cvc.uab.cat/people/aldavert/srv_project_semantic_segmentation.html`

modifiée et environ 1000 tombes (issues de 5 images supplémentaires) y sont rajoutées. Dans ces nouvelles conditions plus statistiquement représentatives, il s'avère que l'approche objet a une précision 7% supérieure à celle de l'approche pixel pour un rappel fixé à 0.41[90], comme l'indique le tableau :

	Approche Pixel	Approche Objet
Rappel	0.41	0.41
Précision	0.23	0.3

TABLE 3.2 – Comparaison de l'approche pixel de [3] avec l'approche objet de [123].

Dans le tableau 3.2, on observe une chute significative de performances (cf tableau 3.1) qui est directement provoquée par la proximité de certaines tombes les unes avec les autres dans la nouvelle base de tests. Aucune des approches pixel et objet n'a de résultats utilisables dans une application réelle.

Notons que l'approche objet utilisée n'effectue pas de quantification en mots visuels. Cette quantification est importante car elle permet de regrouper des motifs de même type. Nous avons donc cherché [90] à mettre en place une quantification en sacs de mots visuels à partir de HOG multi-résolutions [123]. Pour cela, après l'extraction des descripteurs HOG pour une résolution donnée, nous n'effectuons pas de concaténation mais une quantification à partir d'un dictionnaire appris préalablement.

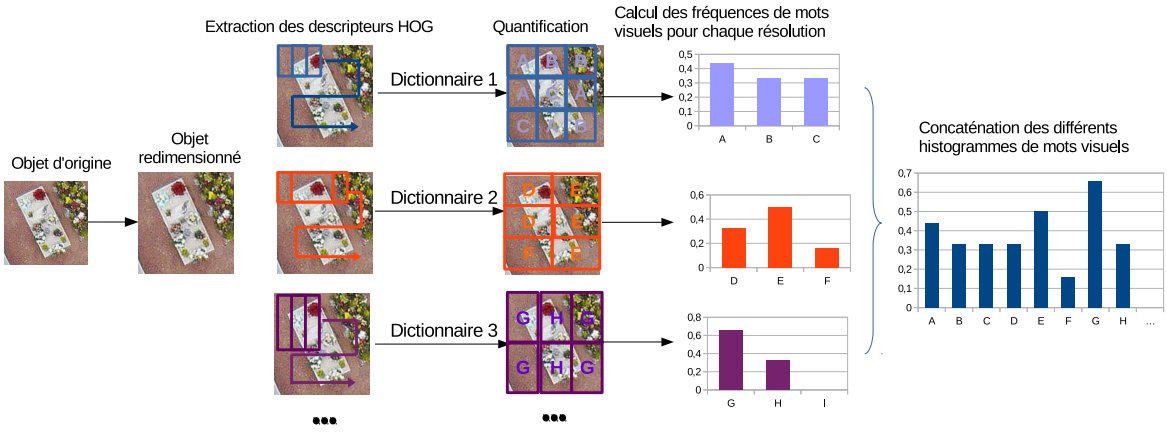


FIGURE 3.5 – Amélioration du descripteur HOG multi-résolutions en utilisant des sacs de mots visuels.

L'illustration 3.5 résume le protocole d'extraction de ce nouveau descripteur. Celui-ci présente deux améliorations majeures. La première consiste en la recherche et l'utilisation pour la quantification de motifs propres à chaque résolution. Ainsi il est envisageable que les motifs de haute résolution représentent des coins, des bords ou des blocs de textures de petites tailles alors que les motifs de basse résolution représentent des croix, ou de longues

lignes de bordure, etc... Le second atout est une uniformisation des dimensions représentant chaque résolution de HOG. En effet, lorsque la fenêtre glissante d'extraction est de petite taille, le nombre de HOG extrait et concaténé est beaucoup plus important que lorsque la fenêtre d'extraction est de grande taille. Grâce à notre descripteur, la taille du vecteur caractéristique issu de chaque dimension correspond à la taille du dictionnaire choisi. Cette taille est fixée permettant de considérer de façon équivalente chaque résolution. Il serait d'ailleurs intéressant de faire varier la taille des dictionnaires en fonction de l'importance de chaque résolution d'analyse.

Nous avons comparé sur la figure 3.6 les performances de l'approche utilisant notre descripteur et celles de l'approche classique provenant de [123]. Comparé aux résultats du précédent stage utilisant l'approche pixel, notre descripteur a un gain de 21% en précision pour un rappel fixé à 41%. De plus, nous constatons que notre descripteur a une précision d'environ 9% supérieure à l'approche de [123] quel que soit le rappel supérieur à 20%.

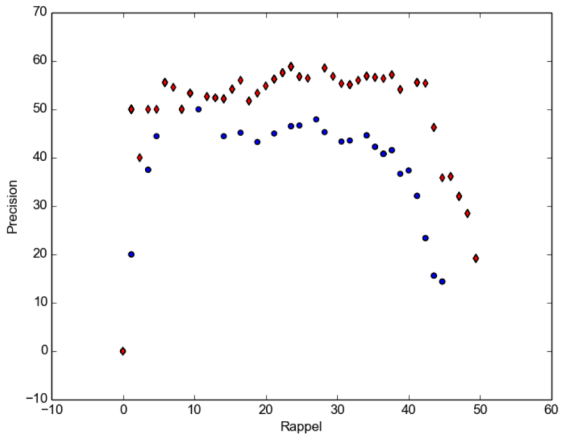


FIGURE 3.6 – Courbe ROC qui représente en bleu les performances de la méthode de Zhu *et al. [123]* et en rouge celles de notre descripteur.

3.4.3 Bilan de la comparaison des approches pixel et objet

Les tombes étant collées les unes aux autres la segmentation pixel les regroupe par paquets ce qui introduit un fort taux d'erreur. Comme il est montré dans [71], les approches objet sont moins sensibles à ce problème. Pour cette raison, nous allons principalement nous intéresser à améliorer les approches objet.

Nous avons vu que l'utilisation de sacs de mots visuels du descripteur HOG permet d'améliorer les performances. Lors de la phase de tests, il est nécessaire d'effectuer une recherche du plus proche voisin afin de quantifier les descripteurs HOG en mots visuels, ce qui est coûteux. Dans la suite de la

thèse, nous continuerons d'utiliser les descripteurs HOG mais en les exploitant différemment sans utiliser de mots visuels.

Il est diffcile d'affirmer que ce descripteur est meilleur qu'un autre sur une autre base d'apprentissage sans effectuer de nombreux tests. Le choix du descripteur induit un classifieur particulier et reste empirique. Les méthodes d'apprentissage profond (*Deep Learning*) permettent d'effectuer l'extraction de caractéristiques et la classification d'objets en simultané. Pour cela, des réseaux vont extraire les meilleurs descripteurs en fonction du problème de classification. Ce processus permet de s'affranchir du choix du descripteur. Dans la section suivante nous présenterons les structures que nous avons améliorées dans la partie contributions.

Chapitre 4

Réseaux de neurones profonds

4.1 Introduction

Les méthodes de classification supervisée abordées précédemment sont divisibles en deux phases. La première est une phase d'extraction de caractéristiques durant laquelle on extrait pour chaque objet de la base d'apprentissage les propriétés discriminantes. Puis, la seconde est une phase d'apprentissage où un classifieur apprend les meilleures frontières entre les différentes classes. Cependant le type de caractéristiques extraites n'est pas forcément optimal pour le problème traité et est choisi indépendamment du classifieur. Une solution pour éviter ce problème est d'effectuer un apprentissage profond (*Deep Learning*) où les phases d'extraction de caractéristiques et de classification sont jointes et se réalisent simultanément.

Le succès de l'apprentissage profond s'explique par ses records dans différentes compétitions. Sur la base "Stanford background dataset", les méthodes profondes récurrentes montrent d'excellentes performances [93] pour la reconnaissance de scènes. Ji *et al.* [58] ont proposé une méthode d'apprentissage profond performante pour la reconnaissance d'actions sur la base TRECVID [82]. Enfin, lors de la compétition ImageNet [28] les méthodes traitant des problèmes de classification et de localisation d'objets sur les images ont déjà fait leurs preuves [65]. Par ailleurs, les réseaux profonds montrent de bonnes perspectives de recherche dans le domaine de la sécurité et plus particulièrement de la stéganographie [92, 91].

En traitement du signal et plus particulièrement en traitement de l'image, les réseaux ayant les meilleures performances sont des réseaux de neurones convolutifs (CNN). En 2006, Hinton [52] en utilisant des réseaux de neurones convolutifs met en avant un mécanisme de représentation de données similaires à celui observable chez l'humain. Pour cela, de façon analogue à un réseau de neurones classique, le signal ou l'image vont être transmis à une couche d'entrée effectuant des transformations. Puis, les résultats transformés vont être transmis à une seconde couche modifiant également les données. Ce processus est répété

sur une succession de couches qui vont s'agencer de manière séquentielle afin qu'une couche au rang n soit entièrement connectée à la couche $n-1$. L'ensemble de ces couches forment un réseau neuronal. Chacune des couches permet d'obtenir une représentation des données avec un certain niveau *d'abstraction*. D'une façon générale, plus la couche est profonde, c'est à dire n est grand, plus le niveau d'abstraction est élevé. Lorsque le niveau d'abstraction est suffisamment élevé il permet de reconnaître des concepts de plus en plus complexes. Par exemple, sur une image, les premières couches servent à reconnaître des contours dans une direction donnée et les couches profondes des groupements de contours formant des croix, rectangles, etc...

Lors de la phase d'apprentissage, les réseaux profonds déterminent par un processus coûteux, appelé rétro-propagation que nous détaillerons dans la section 4.2.1, les valeurs optimales de millions de paramètres. Ce processus est possible à l'aide de calculs déportés sur une ou plusieurs cartes graphiques où chaque carte possède des milliers de cœurs. Plusieurs études tendent à montrer que l'augmentation de la taille des réseaux permet d'améliorer de façon significative les performances [107, 50]. La communauté d'apprentissage profond est donc particulièrement attentive aux apparitions des nouvelles cartes graphiques. D'ailleurs les constructeurs en proposent certaines véritablement dédiées aux calculs pour les réseaux profonds.

Afin d'introduire les méthodes d'apprentissage profond nous expliquerons dans un premier temps le fonctionnement d'un réseau de neurones et les principaux processus associés. Puis, nous traiterons plus spécifiquement des réseaux de neurones convolutifs en expliquant les différents types de couches le composant. Enfin, nous décrirons comment les réseaux profonds peuvent être utilisés afin d'extraire des caractéristiques dites *intelligentes* et de permettre une classification plus performante à l'aide d'un second classifieur tel qu'un SVM.

4.2 Réseaux de neurones

4.2.1 Fonctionnement d'un réseau de neurones

Un réseau de neurones est constitué d'un grand nombre de neurones organisées en couches. Un neurone [96] est un objet qui reçoit un signal et lors d'une phase, dite d'activation [1], émet un nouveau signal modifié dépendant du précèdent. Lors de ce processus, nous pouvons différencier deux types de neurones. Les neurones *intelligents* sont des neurones contenant un poids qui paramètre les opérations effectuées lors de l'activation. Les neurones de *transi-*

1. Par analogie avec l'activation d'un neurone, l'activation du réseau consiste à activer un à un les neurones en commençant par ceux sur les couches d'entrées vers ceux des couches de sorties.

tion sont ceux effectuant un calcul ne dépendant d'aucun poids, il s'agit alors de neurones modifiant un signal suivant une loi constante.

Une couche est une agrégation de neurones possédant tous les mêmes propriétés, c'est-à-dire recevant et émettant des signaux de même nature. Chaque couche prend en entrée une ou plusieurs couches de rang inférieur. Nous noterons $p_i^{(c)}$ le neurone numéro i dans la couche c, avec $i \in \{0, N^{(c)}\}$ et $N^{(c)} - 1$ le nombre de neurones dans la couche c. La couche c est dite *cachée* si elle est directement connectée à la couche $c-1$. Les neurones de la couche c prendront en entrée la concaténation de toutes les sorties des neurones appartenant à la couche $c-1$. Si la couche c n'est connectée à aucune autre couche il s'agit d'une couche *d'entrée* prenant en entrée les données passées au réseau. Si une couche n'est utilisée par aucune autre couche, il s'agit d'une couche de *sortie*. Dans chacun de ces cas le signal reçu par la couche forme le *vecteur d'entrée*. Un réseau peut s'organiser de deux façons différentes :

— une façon cyclique, on parle alors de réseaux récurrents. Il s'agit alors de réseaux où les couches se connectent formant au moins un cycle.

— Une façon acyclique, il s'agit des réseaux les plus courants, le résultat d'une couche de rang n est uniquement transmis à une ou plusieurs couches de rang $(n+k)$ avec $k \in \mathbb{N}_+^*$.

Le neurone classique est un neurone intelligent effectuant un produit scalaire entre son vecteur d'entrée et son poids [95]. A ce produit scalaire nous ajoutons un biais permettant de réguler le seuil d'activation du neurone. Si nous notons $\mathbf{w}_i^{(c)}$, $\mathbf{x}_i^{(c)}$, et $b_i^{(c)}$, le vecteur poids, le vecteur d'entrée et le biais du neurone $p_i^{(c)}$, alors la sortie du neurone, notée $o_i^{(c)}$ est :

$$o_i^{(c)} = \mathbf{w}_i^{(c)}.\mathbf{x}_i^{(c)} + b_i^{(c)} \tag{4.1}$$

Afin que le réseau puisse résoudre des problèmes non linéaires, il est nécessaire que les neurones modifient leurs sorties à l'aide d'une fonction non linéaire dite d'activation, notée ϕ. Ainsi nous obtenons un score noté $s_i^{(c)} \in \mathbb{R}$, tel que :

$$s_i^{(c)} = \phi(o_i^{(c)}) \tag{4.2}$$

Le neurone peut donc se schématiser comme sur la figure 4.1. Ainsi lors de la phase dite d'activation d'une couche c, chaque neurone de cette couche va effectuer un produit scalaire et appliquer une fonction d'activation.

Le résultat des couches de sortie n'est utilisé par aucune autre couche et leur but est de retourner une probabilité d'appartenance à chacune des classes. De façon classique la taille $N^{(c_s)}$ d'une couche de sortie c_s est égale au nombre de classes présentes dans la base d'apprentissage. Ainsi à chaque neurone de cette couche est associé une étiquette. Afin de connaître la probabilité qu'un objet soit d'un certain type, il est nécessaire de normaliser les neurones de cette couche

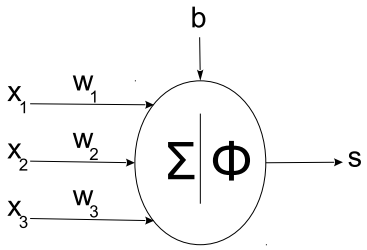


FIGURE 4.1 – Représentation d'un neurone avec x_i et w_i les composantes respectives du vecteur d'entrée et du vecteur poids.

pour obtenir des probabilités. Il est alors classique d'utiliser une fonction *softmax* comme fonction d'activation qui est définie dans l'équation 4.3.

$$s_i = \frac{exp(o_i^{(c_s)})}{\sum_{j=0}^{N^{(c)}} exp(o_j^{(c_s)})} \tag{4.3}$$

Au début de la phase d'apprentissage, tous les poids des neurones intelligents sont initialisés aléatoirement en suivant une distribution donnée. Différents types de distributions peuvent être utilisés comme la distribution gaussienne, uniforme, Xavier [45], etc... Dans le cas de la loi Xavier, de son vrai nom *normalized initialization*, il s'agit d'une initialisation uniforme où chaque couche va calculer les bornes maximale et minimale optimales. Pour cela elle considère le nombre de connexions entrantes, notée n_e, et sortantes, notée n_s, d'un neurone i. Les poids du neurone i sont initialisés selon la loi $U[-\sqrt{\frac{6}{n_e+n_s}}, \sqrt{\frac{6}{n_e+n_s}}]$ avec U une distribution uniforme.

Une fois l'initialisation complète la phase d'apprentissage va chercher à modifier les poids des neurones intelligents pour obtenir la meilleure classification possible sur la base d'entraînement. Ce processus s'effectue à l'aide d'une descente de gradient [97]. Lors de l'activation du réseau, un vecteur va être transmis aux couches d'entrées du réseau. Celui-ci est associé à un vecteur label où t_i est l'étiquette associée au neurone i d'une couche de sortie. Il est alors possible de calculer l'erreur quadratique sur une couche de sortie c suivant l'équation 4.4.

$$E = \frac{1}{2}\sum_{i=0}^{N^{(c)}} (t_i - s_i^{(c)})^2 \tag{4.4}$$

La descente de gradient va donc chercher à minimiser l'erreur E en fonction des paramètres intelligents, c'est à dire les poids (et biais) $\mathbf{w}$. Afin de simplifier les notations, nous notons $w_{ij} \in \mathbb{R}$ le poids reliant le neurone $p_i^{(c_1)}$ à la sortie du neurone $p_j^{(c_2)}$ avec c_1 la couche prenant en entrée c_2, voir le schéma récapitulatif 4.2. Lors de la rétro-propagation d'erreur l'objectif est de modifier chaque poids w_{ij}, par une valeur de correction, notée Δw_{ij}. La valeur de correc-

tion est choisie de telle façon à ce que le poids minimise l'erreur E et suit donc la loi $\Delta w_{ij} \sim -\frac{\delta E}{\delta w_{ij}}$.

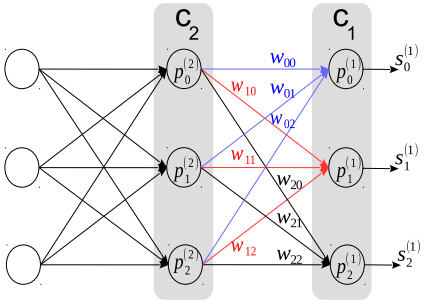


FIGURE 4.2 – Schéma reprenant les notations d'un réseau à deux couches.

Nous exprimons la dérivé partielle comme étant :

$$\frac{\delta E}{\delta w_{ij}} = \underbrace{\frac{\delta E}{\delta s_j^{(c_1)}}}_{C} \underbrace{\frac{\delta s_j^{(c_1)}}{\delta o_j^{(c_1)}}}_{B} \underbrace{\frac{\delta o_j^{(c_1)}}{\delta w_{ij}}}_{A}$$

Pour le terme A :

$$\frac{\delta o_j^{(c_1)}}{\delta w_{ij}} = \frac{\delta(\mathbf{w}_j^{(c_1)}.\mathbf{x}_j^{(c_1)} + \mathbf{b}_j^{(c_1)})}{\delta w_{ij}} = \frac{\delta(\sum_{k=0}^{N^{(c_2)}} w_{kj}.s_k^{(c_2)})}{\delta w_{ij}} = s_i^{(c_2)}$$

Pour le terme B :

$$\frac{\delta s_j^{(c_1)}}{\delta o_j^{(c_1)}} = \frac{\delta(\phi(o_j^{(c_1)}))}{\delta o_j^{(c_1)}} = \phi'(o_j^{(c_1)})$$

ϕ' est appelée fonction de transfert, elle permet de déterminer la force de la correction de l'erreur.

Concernant le terme C, il est nécessaire de différencier deux cas, à savoir celui où le neurone appartient à une couche de sortie et le cas contraire.

Si la couche c_1 est une couche de sortie :

$$\frac{\delta E}{\delta s_j^{(c_1)}} = \frac{\delta \frac{1}{2} \sum_{i=0}^{N^{(c_1)}} (t_i - s_i^{(c_1)})^2}{\delta s_j^{(c_1)}} = -(t_j - s_j^{(c_1)})$$

Si la couche c_1 n'est pas une couche de sortie :

$$\frac{\delta E}{\delta s_j^{(c_1)}} = \sum_{c \in F(c_1)} \sum_{k=0}^{N^{(c)}} \frac{\delta E}{\delta o_k^{(c)}} \frac{\delta o_k^{(c)}}{\delta s_j^{(c_1)}}$$

Avec $F(c_1)$ les couches filles de la couche c_1, c'est à dire celles prenant en entrée la sortie de la couche c_1.

Comme $\frac{\delta o_k^{(c)}}{\delta s_j^{(c_1)}} = \frac{\delta(\mathbf{w}_k^{(c)}.\mathbf{x}_k^{(c)} + \mathbf{b}_k^{(c)})}{\delta s_j^{(c_1)}} = w_{kj}$ nous obtenons :

$$\frac{\delta E}{\delta o_k^{(c)}} \frac{\delta o_k^{(c)}}{\delta s_j^{(c_1)}} = \underbrace{\frac{\delta E}{\delta s_k^{(c)}} \frac{\delta s_k^{(c)}}{\delta o_k^{(c)}}}_{A.B} w_{kj}$$

On reconnaît ici l'apparition des termes $A \times B$, on note $\delta_k^{(c)}$ l'erreur attachée au neurone $p_k^{(c)}$, tel que $\delta_k^{(c)} = \frac{\delta E}{\delta s_k^{(c)}} \frac{\delta s_k^{(c)}}{\delta o_k^{(c)}}$.

En regroupant les différent termes A, B et C on obtient l'équation générique suivante :

$$\frac{\delta E}{\delta w_{ij}} = \delta_j^{(c_1)}.s_i^{(c_2)}$$

Avec

$$\delta_i^{(c)} = \phi'(o_i^{(c)}).\begin{cases} (s_i^{(c)} - t_i) & \text{Si } c \text{ est une couche de sortie} \\ \sum\limits_{c' \in F(c)} \sum\limits_{k=0}^{N^{(c')}} \delta_k^{(c')}.w_{ki} & \text{sinon} \end{cases}$$

Pour éviter les variations de poids trop brusques, pouvant être introduites par des points aberrants et autres bruits, le facteur de correction $\Delta \mathbf{w}_{ij}$ est multiplié par une valeur appelée le coefficient d'apprentissage. Le coefficient d'apprentissage, noté $\epsilon \in [0..1]$, va diminuer au cours de l'apprentissage afin de permettre une convergence vers un minimum local. Cependant, la rétro-propagation ainsi définie peut facilement trouver un minimum local et ne jamais converger vers le minimum global recherché. Pour éviter ce problème, il est possible d'ajouter un terme d'inertie, qui permet de mémoriser la direction du gradient. Ainsi, lorsqu'une variation est introduite, l'inertie est partiellement conservée. Le moment d'inertie, noté $\lambda \in [0..1]$, indique l'importance de la direction du gradient par rapport à la mise à jour à effectuer. A l'apprentissage de chaque échantillon, les poids se mettent à jour suivant l'équation 4.5.

$$\Delta w_{ij} = \lambda \Delta w_{ij} - \epsilon \frac{\delta E}{\delta w_{ij}}$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{4.5}$$

4.2.2 De la non-linéarité avec les fonctions d'activation

Pour que le réseau soit capable de résoudre des problèmes non linéaires il est nécessaire d'introduire de la non linéarité. Pour cela, la fonction ϕ doit être non linéaire et sera utilisée lors de l'activation, voir l'équation 4.2. Historiquement, la première fonction non linéaire introduite comparait la valeur du produit scalaire de l'équation 4.1 avec un seuil [74]. Le biais servant de seuil, la comparaison peut se faire par rapport à 0. Ainsi la première fonction d'activation fut une fonction binaire de la forme : $\phi(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$.

La fonction binaire proposée quantifie le signal avec une perte d'informations, pouvant être potentiellement grande. Aussi de nombreuses autres fonctions ont été proposées afin de transformer et de borner le signal. Parmi ces

fonctions, nous notons les fonctions de type sigmoïde : $\phi(x) = \frac{1}{1+e^{-x}}$, tangente hyperbolique, softsig : $\phi(x) = \frac{1}{1+|x|}$, etc qui sont très utilisées. Dans [45] Glorot et Bengio comparent ces différentes fonctions et montrent que les fonctions tangente hyperbolique et softsig ont des performances très proches. Cependant l'ensemble de ces fonctions et de leurs dérivées sont "coûteux" à calculer. Notons cependant que la dérivée peut souvent s'exprimer en fonction de la fonction d'activation, par exemple, dans le cas de la sigmoïde $\phi'(x) = (1 - \phi(x)).\phi(x)$.

En 2010 Nair et Hinton [79] proposent l'utilisation d'une fonction linéaire rectifiée (Rectified Linear Units ou ReLU) pour réduire les coûts calculatoires et le temps de divergence. Cette fonction est définie comme étant $\phi(x) = max(0, x)$ et sa fonction de transfert est $\phi'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$. Cette fonction est aujourd'hui très utilisée. Dans [65] les auteurs montrent que le ReLU converge 6 fois plus vite que la tangente hyperbolique. Les temps de convergence réduits sont provoqués par la fonction de transfert du ReLU qui, soit transmet l'erreur dans sa totalité, soit transfère une erreur nulle.

Depuis 2010, de nombreuses fonctions d'activation basées sur le ReLU ont été proposées, chacune permettant d'augmenter sensiblement les performances [49, 19]. Ces fonctions utilisent notamment des poids *intelligents* se mettant à jour par rétro-propagation. Notons l'apparition de la fonction SReLU [60] qui est la combinaison de plusieurs fonctions linéaires. Elle s'exprime de la façon suivante :

$$\phi(x) = \begin{cases} t^r + a^r(x - t^r) & x \geq t^r \\ x & t^r > x > t^l \\ t^l + a^l(x - t^l) & x \leq t^l \end{cases}$$

avec t^r, a^r, t^l, a^l quatre paramètres se calculant par rétro-propagation. La fonction SReLU procure un gain allant jusqu'à 2.8% [60] en précision par rapport au ReLU lors de tests sur de nombreuses bases de données : CIFAR, MNIST, ImageNet... Notons que les paramètres se calculant par rétro-propagation augmentent le coût en mémoire mais restent négligeables en coût calculatoire.

4.2.3 Réseaux de neurones stochastiques

Le réseau de neurones présenté dans la partie précédente se met à jour après chaque itération sur un élément de la base d'apprentissage, on parle d'apprentissage en ligne (*on-line*). Afin de converger il est nécessaire d'effectuer plusieurs passes sur la totalité de la base d'apprentissage. Pour chaque échantillon de la base, les poids sont mis à jour ce qui représente un nombre important de calculs.

De plus, d'un élément à l'autre, des fluctuations provoquées par la diversité des échantillons ou des points aberrants peuvent provoquer d'importantes mises à jour et ralentir (ou empêcher) de façon considérable le temps de convergence.

Afin de s'affranchir du problème de fluctuation, l'approche stochastique calcule les corrections des poids pour la totalité de la base d'apprentissage. Elle effectue ensuite une seule mise à jour en prenant en considération la correction moyenne des poids [13]. L'utilisation de ce type de descente de gradient diminue le nombre de mises à jour nécessaire avant la convergence. En effet, lors d'un passage stochastique, l'erreur étant lissée sur la totalité de la base d'apprentissage, la majeure partie des fluctuations disparaît. Cependant sur des bases de grande dimension, parcourir la totalité des échantillons pour n'effectuer qu'une seule mise à jour devient un processus trop coûteux.

Une solution intermédiaire, couramment utilisée dans les architectures logicielles récentes, consiste à utiliser des mises à jour par paquets (mini-batch). Un paquet est un regroupement de k éléments de la base d'apprentissage. L'erreur se calcule uniquement sur les éléments contenus dans le paquet. Après le calcul de l'erreur, une mise à jour est effectuée. Deux cas particuliers existent : si $k = 1$, nous sommes dans le cas d'apprentissage en ligne ; si k est égale à la taille de la base alors nous sommes dans le cas stochastique. Les avantages de cette méthode sont de lisser l'erreur sur des paquets de petites tailles ce qui permet d'effectuer des mises à jour régulièrement. Un avantage non négligeable, propre aux approches stochastiques, est la facilité à la parallélisation GPU. En effet, lors de l'activation d'un neurone ou d'une couche, il est plus facile d'effectuer k fois une opération que de paralléliser l'opération elle même.

Lors de l'apprentissage, afin d'évaluer les performances de notre modèle, une base de validation est utilisée. Une base de validation est un sous-ensemble de la base d'entraînement ne servant pas à l'apprentissage. Cette évaluation permet de stopper le processus de rétro-propagation lorsque le point de convergence est atteint, c'est-à-dire lorsque l'erreur ne diminue plus au court du temps et fluctue. De plus, elle permet en stoppant l'apprentissage d'éviter le sur-apprentissage. Celui-ci est provoqué quand le réseau a tellement de paramètres qu'il apprend tous les cas spécifiques de la base d'apprentissage sans généraliser les concepts.

4.3 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (CNN) sont directement inspirés de l'organisation des neurones dans le cerveau animal pour traiter l'information visuelle [72]. Au sein de ce type de réseaux l'information spatiale est divisée en régions. Chacune des régions pouvant se chevaucher et être traitée par un ou plusieurs neurones. Lors de l'apprentissage automatique, l'information spatiale

va être traitée par une succession de filtres et d'opérations permettant d'extraire des motifs. Ces filtres et ces transformations sont issues de couches de convolutions, de couches de sous-échantillonnage et de couches de normalisation comme l'indique la figure 4.3. La dernière couche spatialisée est ensuite transmise à une ou plusieurs couches entièrement connectées. Dans les sous-sections suivantes nous décrirons les différents types d'opérations possibles.

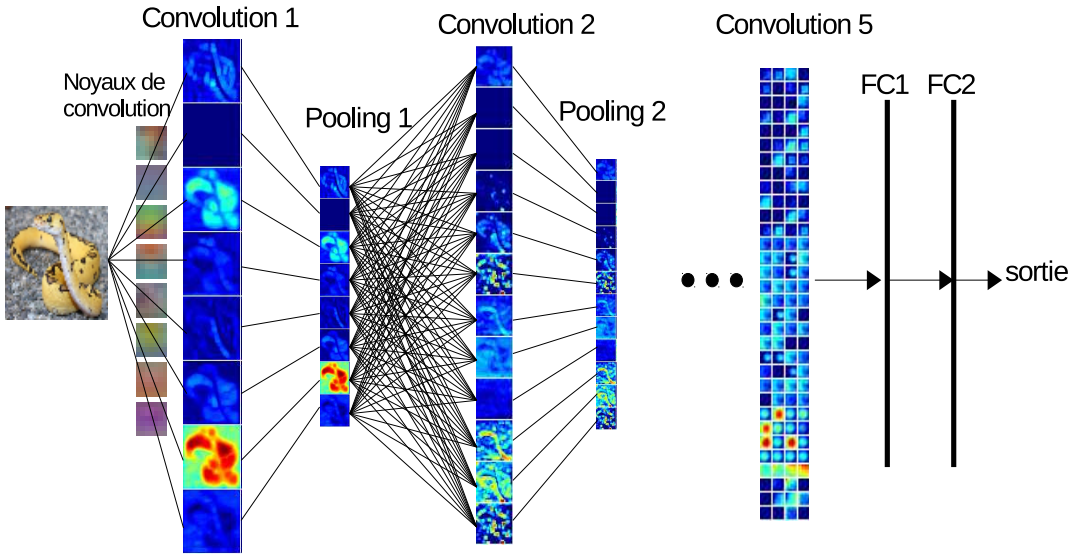


FIGURE 4.3 – Exemple d'un réseau de convolutions avec deux couches de convolutions et de sous-échantillonnage représentées et connectées à deux couches entièrement connectées (FC1 et FC2).

4.3.1 Couche de convolutions

Afin d'expliquer la convolution effectuée dans le cas du CNN, pour générer des cartes de caractéristiques, il convient d'expliquer la convolution 2D dans le cas général. Si nous considérons un noyau $\mathbf{K} \in \{0...w\} \times \{0..h\}$ et une image $I \in \mathbb{R}^2$, l'image I convoluée par $\mathbf{K}$ se note $I * \mathbf{K}$, et est définie comme :

$$(I * \mathbf{K})_{x,y} = \sum_{x'=0}^{w} \sum_{y'=0}^{h} \mathbf{K}_{x',y'} . I_{x+x'-\frac{w}{2},y+y'-\frac{h}{2}}$$

Les couches de convolutions sont des couches contenant des neurones de type convolutif. Chaque neurone convolutif applique une somme de convolutions 2D à un signal d'entrée afin d'y extraire des motifs et des caractéristiques [55]. Nous expliquons leur fonctionnement dans le cas du traitement d'une image multi-canaux (largeur, hauteur et nombre de canaux) mais cela reste facilement extensible à des applications 2D (audio) et à d'autres traitements.

Nous considérons que le neurone $p_i^{(c)}$ possède un poids, aussi appelé noyau de convolutions, noté $\mathbf{w}_i^{(c)} \in \mathbb{R}^3$. Les deux premières dimensions de ce noyau représentent la largeur et la hauteur du filtre de convolutions. La troisième di-

mension représente le canal sur lequel la convolution est appliquée. Nous notons $\mathbf{w}_i^{(c)}(p)$ le noyau 2D traitant le canal p.

Pour un réseau CNN, comme pour un réseau de neurones classiques, un neurone prend en paramètre toutes les sorties des couches qui y sont connectées. Cependant, contrairement aux couches entièrement connectées, dans un CNN la sortie des neurones convolutifs est une image et non un scalaire. Il convient alors de redéfinir les équations 4.2 et 4.1. Si nous considérons que la couche (c) prend uniquement en paramètre les sorties de la couche $(c-1)$, la sortie $\mathbf{o}_i^{(c)}$ du neurone convolutif se calcule comme :

$$\mathbf{o}_i^{(c)} = b_i^{(c)} + \sum_{j=0}^{N^{(c-1)}} (\mathbf{w}_i^{(c)}(j) * \mathbf{s}_j^{(c-1)}) \tag{4.6}$$

avec $b_i^{(c)}$ le biais du neurone convolutif, c'est à dire une constante s'ajoutant à toutes les positions de l'image résultante.

Il convient d'introduire la non linéarité en appliquant une fonction ϕ telle que :

$$\mathbf{s}_i^{(c)} = \phi(\mathbf{o}_i^{(c)})$$

L'image résultante $\mathbf{s}_i^{(c)}$ est appelée une carte de caractéristiques (*feature map*).

Durant l'activation d'un neurone convolutif, on parle de noyau partagé (*shared weights*) car un même noyau est utilisé pour convoluer toutes les positions des images d'entrées. Le calcul est indépendant de la position dans l'image et permet d'obtenir de nombreuses invariances. Ce fonctionnement est très important pour la reconnaissance d'images où l'on veut reconnaître des motifs selon une même règle quelque soient leurs positions. Durant l'apprentissage, les poids sont mis à jour en calculant la somme des erreurs à toutes les positions où la convolution est appliquée. A partir de cette erreur les poids sont mis à jour selon la même règle que l'équation 4.5.

4.3.2 Couche de sous-échantillonnages

Les couches de sous échantillonnages (*pooling*) servent à regrouper et fusionner les valeurs des cartes de caractéristiques. Cette couche est une couche de *transition* c'est à dire qu'elle ne possède pas de poids à mettre à jour. Son fonctionnement repose sur une fenêtre glissante de taille $w \times h$ qui parcourt chaque carte de caractéristiques avec un pas de chevauchement donné. A chaque position évaluée une opération de quantification attribue une valeur au contenu de la fenêtre glissante.

Deux grands types de sous échantillonnages existent, celui par la moyenne et celui par le maximum. La moyenne permet de considérer toutes les va-

leurs de la carte de caractéristiques pour une fenêtre glissante donnée. Lors de la rétro-propagation du gradient, l'erreur est répartie entre les différentes valeurs de la fenêtre glissante. A l'inverse, le maximum ne conserve que la caractéristique la plus élevée. Aussi, lors de la rétro-propagation du gradient, l'erreur est uniquement propagée sur la position du pixel possédant la plus forte valeur. Il a été montré dans [98] que le sous-échantillonnage par maximum converge plus rapidement lors de la phase d'apprentissage que le sous échantillonnage par la moyenne.

Lors de l'application d'une telle couche il est courant d'utiliser un chevauchement qui permet de diminuer la taille de représentation des cartes de caractéristiques. Le gain calculatoire ainsi obtenu ne peut pas être négligé. Cependant, lors des sous-échantillonnages, une partie de l'information est perdue pouvant réduire de façon significative les performances. Pour cela la tendance est d'utiliser de petits chevauchements ou même d'éviter l'utilisation (abusive) de couches de sous échantillonnages [105].

4.3.3 Couche de normalisation

Les noyaux étant initialisés aléatoirement, il est possible que certaines valeurs de cartes de caractéristiques saturent et donnent des scores trop élevés comparés à ceux des autres noyaux. Il est alors important de normaliser les différentes cartes de caractéristiques entre elles. Pour s'affranchir de ce problème des couches de normalisation, pouvant être de transition ou intelligentes, sont mises en place. Chaque carte de caractéristiques est normalisée en fonction des valeurs des autres cartes de caractéristiques.

S'inspirant directement du fonctionnement de certains neurones biologiques [69], Jarrett *et al.* proposent [55] une normalisation par soustraction de la somme des cartes de caractéristiques convoluées avec un noyau $\mathbf{K} \in \mathbb{R}^{\mathbf{2}}$ gaussien. Ce type de normalisation est appelé normalisation par contraste local. Si nous considérons la couche c normalisant la couche $(c-1)$, la carte de caractéristiques $\mathbf{s}_i^{(c)}$ du neurone $p_i^{(c)}$ est définie dans l'équation 4.7.

$$\mathbf{s}_i^{(c)} = \mathbf{s}_i^{(c-1)} - \sum_{j=0}^{N^{(c)}} \mathbf{K} * \mathbf{s}_i^{(c-1)} \tag{4.7}$$

S'appuyant sur ce fonctionnement, Krizhevsky *et al.* proposent une autre normalisation appelée normalisation par luminosité [65]. Ce type de normalisation ne considère plus le voisinage d'une caractéristique à une position donnée. Nous notons $s_i^{(c-1)}(x,y)$ la valeur de la carte de caractéristiques du neurone $p_i^{(c-1)}$ à la position (x,y).

$$s_i^{(c)}(x,y) = \frac{s_i^{(c-1)}(x,y)}{\left(k + \alpha . \sum_{j=max(0,i-n/2)}^{min(N^{(c)},i+n/2)} \left(s_j^{(c-1)}(x,y)\right)^2\right)^{\beta}} \tag{4.8}$$

avec n le nombre de cartes de caractéristiques à considérer. En effet, il n'est pas nécessaire de considérer tous les noyaux d'une même couche puisque le phénomène de saturation est rare. De plus, ne considérer que n voisins permet de réduire le coût de calculs. Les hyper-paramètres k, α et β sont déterminés à l'aide d'une base de validation [65].

Récemment, un autre type de normalisation a été introduit par Ioffe et Szegedy permettant de régler le problème de saturation des noyaux et d'être robuste aux différentes distributions d'entrée [54]. Cette démarche repose sur une normalisation de chaque caractéristique appartenant à une carte de caractéristiques par rapport à sa statistique propre. Pour cela, il faut considérer que la descente de gradient utilisée est stochastique. Ainsi, pour une caractéristique donnée, à partir des différentes valeurs obtenues sur le paquet, un ensemble de paramètres statistiques est calculé. Ce calcul statistique par paquet a donné son nom à la méthode à savoir la "*batch normalization*" (BN). Lors de la BN on calcule une moyenne et une variance sur chaque valeur des cartes de caractéristiques. Ces deux valeurs statistiques sont moyennées avec l'ensemble des moyennes et variances calculées sur tous les paquets passés à la phase d'apprentissage.

Notons $\mu_i^{(c-1)}(x,y)$ et $\sigma_i^{(c-1)}(x,y)$ la moyenne et la variance moyennées de la carte de caractéristiques $\mathbf{s}_i^{(c-1)}$ à la position (x,y).

Le couche de BN effectue le calcul décrit dans l'équation suivante :

$$s_i^{(c)}(x,y) = \frac{s_i^{(c-1)}(x,y) - \mu_i^{(c-1)}(x,y)}{\sqrt{\sigma_i^{(c-1)}(x,y)}} \tag{4.9}$$

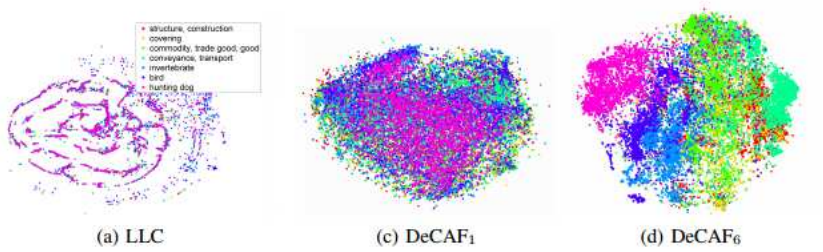
Afin de normaliser les différentes cartes de caractéristiques entre elles, les auteurs de [54] proposent deux hyper paramètres γ et β se mettant à jour par rétro-propagation. Ces hyper paramètres sont calculés par carte de caractéristiques. Le calcul effectif de la BN en considérant γ et β est donné dans l'équation 4.10.

$$s_i^{(c)}(x,y) = \frac{\gamma}{\sqrt{\sigma_i^{(c-1)}(x,y)}} . s_i^{(c-1)}(x,y) + (\beta - \frac{\gamma . \mu_i^{(c-1)}(x,y)}{\sqrt{\sigma_i^{(c-1)}(x,y)}}) \tag{4.10}$$

Les auteurs de [54] montrent que l'utilisation de la BN réduit l'erreur de 1% en utilisant un des réseaux les plus performants sur la base ImageNet [119]. La BN est très utilisée dans les architectures de réseaux récentes [50, 46].

4.4 Utilisation des CNN pour extraire des caractéristiques sophistiquées

Les CNN permettent au sein des couches de convolutions de reconnaître des motifs propres au problème à résoudre. En effet, les noyaux de convolutions sont calculés à partir de la base d'apprentissage. Une représentation des cartes de caractéristiques à l'intérieur des différentes couches du réseau permet de mettre en évidence une représentation spécifique pour un problème donné. En effet, sur l'illustration 4.4, on constate que pour un type de descripteur donné (par exemple *Locality-constrained Linear Coding* ou LLC [117]) la représentation d'un objet ne forme pas une zone homogène et le descripteur ne sépare pas les différentes classes. À l'inverse, la sortie de la $6^{ème}$ couche d'un réseau CNN permet de regrouper les caractéristiques d'un même objet entre elles. Au cours de l'apprentissage les différentes couches de convolutions se spécialisent et forment un descripteur pertinent pour le problème donné.



(a) LLC (c) $DeCAF_1$ (d) $DeCAF_6$

FIGURE 4.4 – Image représentant la projection 2D t-SNE [115] de différents types de vecteurs caractéristiques calculés sur la base de validation ILSVRC-2012. L'image (a) représente des caractéristiques de type LLC [117], on voit que les différentes classes sont mélangées et le classifieur devra réussir à les séparer. Les images (c) et (d) représentent les cartes de caractéristiques d'un réseau CNN à la première couche et à la $6^{ième}$ couche. A la première couche, les différentes classes sont mélangées car le réseau n'a pas créé de descripteur pertinent. Sur la $6^{ième}$ couche, le réseau a un descripteur pertinent permettant de séparer les classes en 2D (figure provenant de [31]).

De plus, les couches de sous-échantillonnages introduisent une robustesse aux translations et rotations. En 2015, il fut montré [122] qu'un nombre suffisamment élevé de couches de convolutions permet au réseau de devenir robuste aux orientations. Ce résultat permet d'expliquer les performances de certaines architectures comme GoogLeNet [107] (le gagnant du challenge ImageNet 2014), VGG [102] et ResNet [50] qui contiennent 22, 19 et 152 couches intelligentes respectivement.

Puisque les couches du réseau permettent une représentation intelligente et robuste, il est alors intéressant d'utiliser cette représentation comme descripteur et d'effectuer une classification objet. Ainsi plusieurs auteurs [111, 14] ont remplacé la dernière couche du réseau (le plus souvent le *softmax*) par un SVM

permettant de réduire l'erreur de plus de 2% sur la base CIFAR. Il est également intéressant d'utiliser, après l'apprentissage, les cartes de caractéristiques de certaines couches pour effectuer l'apprentissage d'un SVM [121, 122].

Afin de trouver les meilleures caractéristiques à extraire pour l'apprentissage d'un SVM il est possible d'effectuer un apprentissage non supervisé. Pour cela, nous pouvons mettre en place une architecture de type *auto-encoder*. Il s'agit d'une architecture où l'entrée du réseau est également la sortie attendue [52]. Ainsi les différentes couches du réseau permettent d'obtenir une représentation réduite des données. Les caractéristiques obtenues par par la partie codeur présentent plusieurs robustesses ce qui les rend très utilisées dans les problèmes de classification [8, 68]. Ces représentations peuvent être utilisées comme descripteur pour effectuer l'apprentissage d'un autre classifieur, comme le SVM [61] ou des forêts d'arbres aléatoires [63].

4.5 Réduction des temps de calculs

Afin de localiser les objets contenus dans une image le concept de fenêtres glissantes testant toutes les positions est traditionnellement utilisé. L'activation d'un réseau pour une fenêtre de taille réduite (64×64 pixels) est rapide sur GPU mais répéter cela pour des millions de positions de la fenêtre glissante est très coûteux. Ce problème est également présent en traitement et localisation d'objets sur vidéos [6]. De nombreuses recherches cherchent à réduire le coût calculatoire de la phase d'évaluation [6, 5]. Il est important de considérer le compromis entre gain en temps de calculs et perte en efficacité. En effet, des méthodes comme *WordChannel* traite une image en 0.06 seconde mais ne détecte pas 42% des objets. Sur la même base de données et avec la même méthodologie, LFOV-2St [5] traite une image en 0.55 seconde et détecte 7% plus d'objets.

Récemment Angelova *et al.* [6] ont proposé une architecture nommée *DeepCascade* permettant d'obtenir un bon compromis entre la vitesse de traitement de les performances. La figure 4.5 montre que l'approche *DeepCascade* est à la fois plus rapide et plus performante que la plupart des autres méthodes pour la détection de piétons dans la base de données de référence [30].

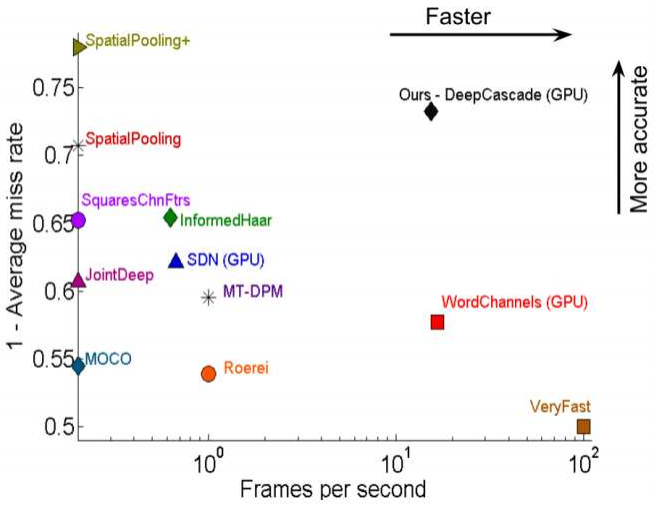


FIGURE 4.5 – Comparaison des performances et des vitesses de traitement de différents algorithmes (Image provenant de [6]).

Dans l'approche de *DeepCascade*, les auteurs entraînent un réseau de très petite taille, composé uniquement de 4 couches intelligentes. Ce réseau fut utilisé en premier dans l'article [5]. L'objectif de ce réseau est d'éliminer les fenêtres candidats où la probabilité d'un objet d'intérêt est très faible. Les fenêtres possédant une probabilité suffisamment élevée passent ce premier filtre et sont soumises à un réseau de plus grande taille. Ce réseau est identique à celui présenté par Krizhevsky *et al.* [65] avec des dimensions de cartes de caractéristiques adaptées au problème de la détection de piétons à savoir 64x64. Le rajout simple de ce réseau de petite taille permet une accélération d'un facteur 80 lors du passage de la fenêtre glissante.

Afin de rendre l'approche *DeepCascade* plus rapide, avant le passage par le petit réseau, les images sont filtrées par une cascade de classifieurs définie dans [11]. Contrairement à la cascade originale, plus la taille de la cascade est grande moins le rappel est important [12]. L'approche de *DeepCascade* n'utilise que 10% de la cascade ce qui permet un premier filtrage extrêmement rapide sans perdre en rappel.

D'autres méthodes permettent de réduire le coût d'évaluation des approches profondes en s'affranchissant du concept de fenêtres glissantes. Une solution consiste à localiser directement la position des objets sur des images de 'grandes' dimensions. Pour cela, chaque image est partitionnée en une grille plus ou moins dense [94, 24, 44, 108, 34]. Chaque partie de la grille peut s'activer indépendamment des autres si elle contient la totalité ou une partie d'un objet d'intérêt. La figure 4.6 représente le fonctionnement de ce style d'architecture. Dans le contexte de cette thèse, ce genre d'approche est inapproprié. En effet, les objets étant collés les uns aux autres, il est nécessaire d'utiliser des blocs de petite taille. La frontière entre les objets urbains ne sera donc pas correctement

détectée, ce qui provoquera des problèmes analogues à celui de la segmentation pixel.

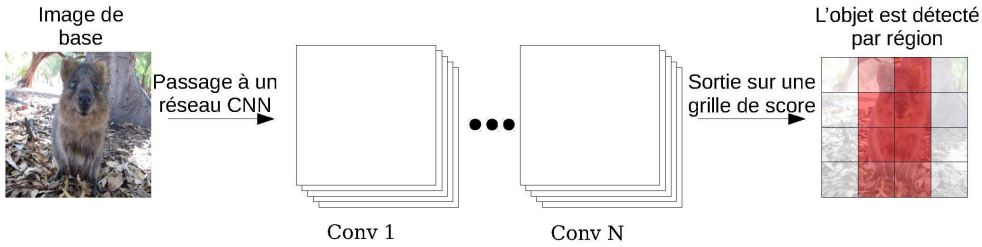


FIGURE 4.6 – Représentation simplifiée d'une architecture par détection de régions. Plus la sortie est rouge, plus la probabilité de présence d'un Quokka est forte. Les frontières de l'objet sont grossièrement détectées.

4.6 Discussions

Dans cette section nous avons détaillé les mécanismes de bases de fonctionnement d'un réseau de neurones convolutifs profond. Nous avons vu que dans le traitement de l'image ils sont composés d'un ensemble de couches de convolutions et de couches de sous-échantillonnages. Toutes ces couches permettent d'extraire des caractéristiques de très haut niveau qui sont transmises à des couches entièrement connectées. Pour une tâche de localisation, des systèmes de filtres peuvent être mis en place en utilisant une cascade de classifieurs et des réseaux plus petits afin d'éliminer le plus de fenêtres candidats possible.

Nous mentionnons particulièrement trois réseaux très connus à savoir les réseaux : AlexNet [65], GoogLeNet [107] et ResNet [50]. Les structures de ces trois réseaux sont décrites dans le tableau 4.1. Le réseau AlexNet a été l'une des premières grosses structures à donner des résultats significatifs lors de la compétition ILSVRC2012. Il fut ensuite dépassé par l'architecture de GoogLeNet, lors de la compétition ILSVRC2014. Nous constatons que la différence entre ces deux architectures est double. D'une part, GoogleLeNet utilise des modules d'inception, c'est-à-dire des blocs de plusieurs filtres sur un même niveau traitant la même information avec différentes résolutions. Ce genre d'architecture induit une forte robustesse aux traitements multi-échelles. D'autre part la taille du réseau est augmentée en rajoutant des couches de convolutions en profondeur. Cependant le réseau GoogLeNet n'a pas plus de paramètres (c'est à dire de poids) à optimiser que le réseau AlexNet. Ce traitement très profond avec un grand nombre de couches permet de traiter toute l'information avec des couches de convolutions et rend l'utilisation de couches entièrement connectées moins utile. En 2015, He *et al.* proposent l'architecture ResNet qui est

composée de plus de 150 couches de convolutions. Cette architecture a comme particularité de sommer les résidus issus des convolutions avec les images d'entrées. L'évolution de ces réseaux tend à supposer que la création de réseaux toujours plus profonds avec des protocoles de normalisation comme la BN ou le *dropout* [106, 54] ne peut qu'augmenter les performances de reconnaissance et de localisation.

Dans notre cas d'étude, nous n'effectuons pas une classification mais une localisation. Il est nécessaire d'adapter ces réseaux performants pour traiter des imagettes de tailles réduites. Dans de nombreux travaux [6, 122, 29], le réseau de AlexNet est utilisé pour détecter les objets à l'aide de fenêtres glissantes. De plus, les réseaux de grandes dimensions contiennent beaucoup de couches de sous-échantillonages ce qui peut, dans le cas d'objets collés, mener à un mauvais recalage de la fenêtre de détection. Notre choix se porte donc vers l'utilisation d'un réseau type AlexNet qui sera détaillé dans le chapitre traitant de nos contributions sur CNN.

AlexNet

Nom couche	Taille des cartes de caractéristiques de sortie	Noyaux (|K|, N(c))
Conv1 Pool	27x27	(11x11, 96)
Conv2 Pool	13x13	(5x5, 256)
Conv3	13x13	(3x3, 384)
Conv4	13x13	(3x3, 384)
Conv5 Pool	7x7	(3x3, 256)
FC		4096
FC		4096

ResNet

Nom Couche bla	Taille des cartes de caractéristiques de sorties	Noyaux (|K|, N(c))	Répétition du noyau
Conv1	112x112	(7x7, 64) Max pooling	
Conv2	56x56	(1x1, 64) (3x3, 64) (1x1, 256)	x3
Conv3	28x28	(1x1, 128) (3x3, 128) (1x1, 512)	x8
Conv4	14x14	(1x1, 256) (3x3, 256) (1x1, 1024)	x36
Conv5	7x7	(1x1, 512) (3x3, 512) (1x1, 2048) Average pooling	x3
FC	1	1000	

GoogleLeNet

Nom Couche	Taille des cartes de caractéristiques de sortie	Filtre 1x1	Filtre 3x3	Filtre 5x5	Filtre 7x7	Pooling max Et Filtre 1x1
Conv1 Pooling	56x56				64	
Conv2	56x56	64				
Conv3 Pooling	28x28		192			
Inception3a	28x28	64	96	16		32
Inception3b Pooling	14x14	128	192	96		64
Inception4a	14x14	192	203	48		64
Inception4b	14x14	128	256	64		64
Inception4c	14x14	112	288	64		64
Inception4d Pooling	8x8	256	320	128		128
Inception5a	8x8	256	320	128		128
Inception5b	8x8	384	384	128		128
Softmax						

TABLE 4.1 – Détails des trois architectures connues qui sont AlexNet, GoogLeNet et ResNet.

Deuxième partie

Contributions

Chapitre 5

Méthodologie utilisée pour l'analyse des résultats

5.1 Localisation des objets et fusion des résultats

Lors d'une approche de classification objet, basée sur le concept de fenêtres glissantes, le modèle attribue un vecteur de probabilité à chaque fenêtre testée. Ce vecteur de probabilités représente la certitude que le classifieur attribue à la fenêtre qu'elle soit un objet d'une certaine classe. Ainsi, le plus souvent, les fenêtres testées sont associées au label correspondant à la probabilité la plus importante. Lorsqu'un objet est trouvé dans la fenêtre glissante, la fenêtre devient ce que l'on appelle une fenêtre englobante (*bounding box*). Cependant, comme nous l'avons décrit dans la partie 3.3, l'image testée est ensuite redimensionnée afin d'être analysée à différentes résolutions. Ce changement de dimension ainsi que le chevauchement de la fenêtre glissante ont pour effet qu'un même objet est recouvert plusieurs fois, de façon presque identique. Dans cette thèse, les objets urbains étudiés, ne s'auto-contiennent pas c'est-à-dire qu'un objet urbain ne peut pas être présent à l'intérieur d'un autre objet urbain. Il est alors nécessaire de fusionner les différentes fenêtres englobantes pour éviter de comptabiliser plusieurs fois un même objet et obtenir la meilleure localisation. Pour cela, nous utilisons le ratio des aires, noté a, utilisé lors de nombreuses compétitions pour l'évaluation [35]. Ce ratio permet de déterminer si deux fenêtres englobantes notées F_1 et F_2 se superposent correctement. Le ratio a est défini par l'aire de la fenêtre englobante intersectant F_1 et F_2 sur l'aire de la fenêtre englobante unissant F_1 et F_2, comme l'indique l'équation :

$$a = \frac{Aire(F_1 \cap F_2)}{Aire(F_1 \cup F_2)} \tag{5.1}$$

Lorsque a est supérieur à un seuil (le plus souvent 0.5), F_1 et F_2 sont considérées comme étant superposées. Il est également possible d'utiliser des critères moins stricts pour la fusion de fenêtres telle que la distance séparant les barycentres de F_1 et F_2.

Lorsque deux fenêtres englobantes sont superposées il est alors nécessaire de les fusionner. Plusieurs solutions existent dans la littérature. La première consiste à fusionner les coordonnées des fenêtres englobantes [100]. La seconde consiste à supprimer toutes les fenêtres englobantes superposées qui ont une probabilité associée moins importante [37, 110].

Dans cette thèse nous ne considérons que le cas de la classification objets et optons pour la stratégie consistant à ne conserver que les fenêtres englobantes associées à la probabilité maximale.

5.2 Évaluation des méthodes de classification

Une fois les fenêtres englobantes fusionnées nous considérerons qu'une fenêtre englobante est correctement positionnée si son ratio d'aire a, avec son objet de référence dans la vérité terrain, est supérieur à un seuil (que l'on peut prendre égal à 0.5 comme dans [35]). Notons que chaque objet de la base de vérité terrain ne doit être détecté qu'une et une seule fois, un objet détecté n fois sera donc comptabilisé $n-1$ fois comme un faux positif et 1 fois comme un vrai positif. Dans la phase de tests, seules les fenêtres englobantes avec une probabilité associée supérieure à un seuil α sont conservées. Si la probabilité est inférieure à la valeur de α, la fenêtre englobante n'est pas considérée dans la détection et n'intervient donc pas dans l'évaluation. Pour évaluer nos méthodes nous calculons le *rappel* et la *précision* en fonction du seuil α, tel que :

$$rappel = \frac{TP}{TP+FN} \text{ et } precision = \frac{TP}{TP+FP}$$

avec TP, FP et FN les nombres de vrais positifs, faux positifs et faux négatifs respectivement.

De plus, nous pouvons calculer la F_{mesure} qui est définie comme la moyenne harmonique du rappel et de la précision :

$$F_{mesure} = 2 \times \frac{rappel \times precision}{rappel + precision}$$

Dans le cas de la localisation de tombes, le nombre de vrai positifs est le nombre de tombes réelle détectées ; le nombre de faux positifs est le nombre de tombes détectées qui ne sont pas des tombes ou qui sont des tombes déjà détectées ; le nombre de faux négatifs est le nombre de tombes non détectées.

L'ensemble des points formés par les valeurs du rappel et de la précision forment la courbe ROC (*Receiver Operating Characteristic*). Dans la suite de ces travaux, nous représentons beaucoup de nos résultats de cette façon. La figure 5.1, donne l'exemple d'une courbe ROC typique. Comme on peut le voir, en général, plus le rappel est élevé (c'est à dire le nombre de tombes détecté est important) moins la précision (c'est à dire le nombre de tombes correctement

trouvées) est grande. A l'inverse plus la précision est grande moins le rappel est élevé. Nous utiliserons beaucoup l'aire sous la courbe ROC pour comparer les différents résultats. Ainsi nous comparons la précision moyenne pour un intervalle de rappel donné.

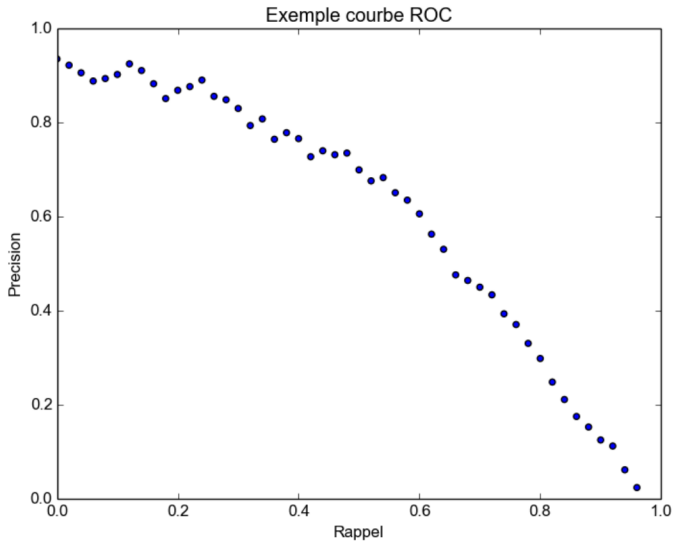


FIGURE 5.1 – Exemple de courbe ROC.

5.3 Présentation de la base de données et de notre protocole d'évaluations

La base de données utilisée est celle présentée dans la section 1.2. Cette base est découpée en trois bases : la base d'apprentissage, la base de validation et la base de tests. La base d'apprentissage est utilisée pour entraîner les algorithmes de classification. La base de validation est utilisée pour valider et optimiser certains paramètres d'apprentissage. Cette base se compose de deux ou trois images afin de contenir entre 600 et 800 tombes. La base de tests est utilisée pour évaluer nos méthodes d'apprentissage. Elle est constituée de deux images dont le nombre total de tombes varie entre 600 et 1000.

Une fois l'apprentissage et l'évaluation effectués, nous redécoupons différemment la base de données et répétons ce processus trois fois. Ce protocole permet d'obtenir trois résultats qui seront moyennés afin d'obtenir un résultat statistiquement significatif. La figure 5.2 donne un aperçu des images de tests utilisées durant ce protocole.

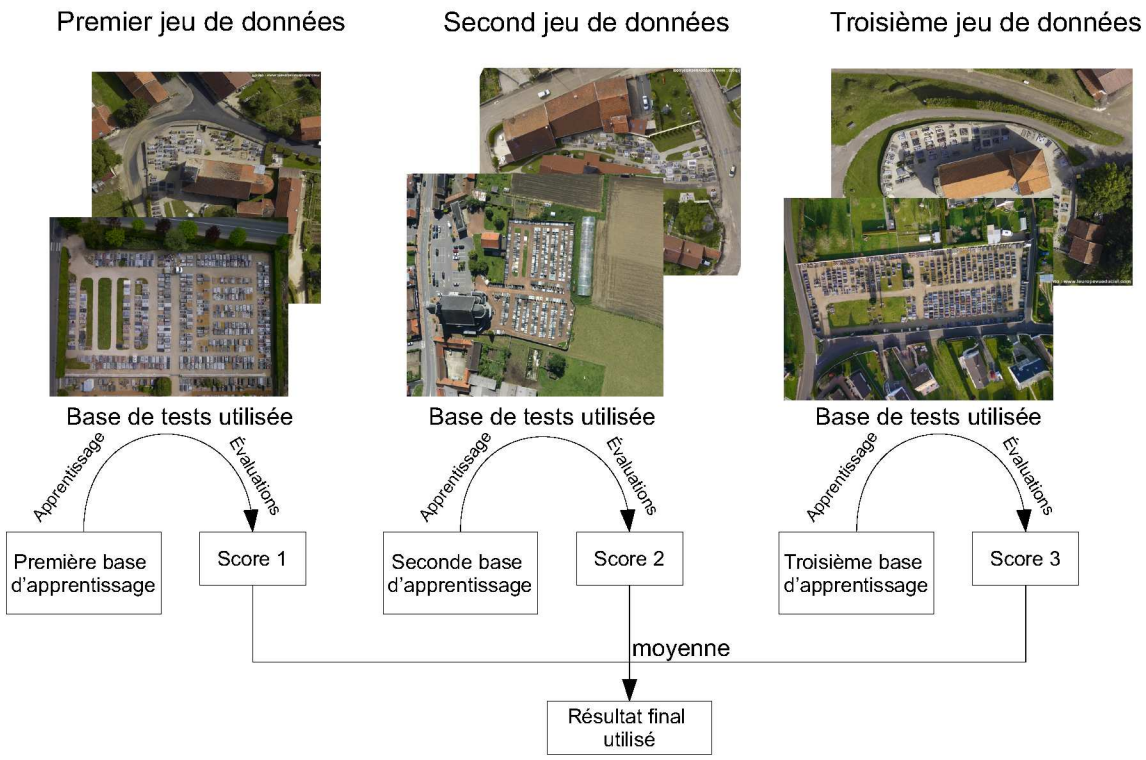


FIGURE 5.2 – Illustration des images de tests utilisées dans chacun des trois jeux de données.

Les ensembles des méthodes proposées et testées reposent sur le phénomène de fenêtres glissantes. Leur taille a un rôle important et doit être proche de la taille moyenne d'une tombe standard. Les dimensions moyennées des tombes étant de 1×2 mètres, pour une résolution de 2.5cm/pixel cela correspond à 40×80 pixels. Différentes tailles d'imagettes ont été testées expérimentalement afin de trouver la meilleure dimension normalisée. Ainsi, nous avons utilisé des tailles d'imagette de 70×70 pixels pour les méthodes basées sur l'extraction des descripteurs HOG et 64×64 pixels pour les méthodes utilisant un CNN.

Pour redimensionner les tombes il est nécessaire d'effectuer une interpolation. A l'aide de la libraire PIL [1] nous avons testé plusieurs interpolations telles que l'interpolation linéaire, bicubic et celle du plus proche voisin. Suite à ces expérimentations l'interpolation bicubic donnent les meilleurs résultats, elle sera alors utilisée dans la suite de nos travaux.

1. Site officiel de PIL : http ://www.pythonware.com/products/pil/

Chapitre 6

Présentation du Réseau de SVM

6.1 Introduction aux problèmes de fusion

Nous avons vu dans la section 2.4.2, que le descripteur HOG est extrait sur chaque imagette des bases d'apprentissages et de tests. Pour être le plus discriminant possible, ce descripteur est extrait sur des fenêtres glissantes de différentes tailles qui correspondent aux différentes résolutions. On notera r une résolution avec $r \in \{0..R\}$ et $R-1$ le nombre de résolutions différentes. Le descripteur HOG, extrait avec une résolution r, est noté $\mathbf{f}^r$. Lors de la création du descripteur HOG final, il est habituel de concaténer l'ensemble des vecteurs extraits aux différentes résolutions [123]. Dans notre cas, notons que la taille du vecteur HOG final est calculée sur 81 résolutions et a une dimension de 34084.

Le nombre de positions testées par la fenêtre glissante est différent en fonction de la résolution. Comme le montre la figure 6.1, à faible résolution le nombre de positions testées par la fenêtre glissante est moins important qu'avec une haute résolution. En effet, pour des fenêtres glissantes de petites tailles, ici

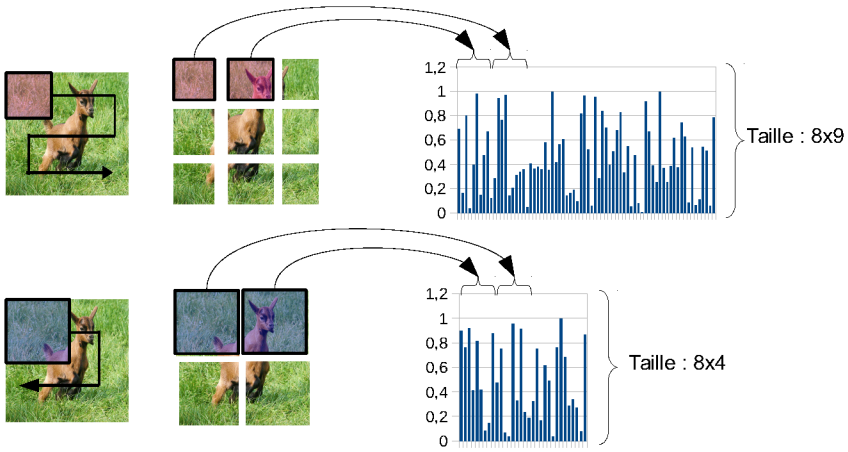


FIGURE 6.1 – Illustration de l'influence de la résolution d'extraction r sur la dimension du vecteur $\mathbf{f}^r$.

un tiers de l'imagette, et en ne considérant aucun chevauchement, la taille du

vecteur HOG pour cette résolution, notée $r = 0$, est de $|\mathbf{f}^0| = 81$. Dans le cas où la fenêtre glissante est plus grande, ici la moitié de l'imagette, c'est-à-dire a une résolution plus faible, notée $r = 1$, la taille du vecteur HOG est $|\mathbf{f}^1| = 36$. Aussi le classifieur recevra beaucoup plus d'informations provenant d'une résolution haute que d'une résolution faible.

De ce fait, lors de la concaténation des différents vecteurs $\mathbf{f}^r$ entre eux, les vecteurs issus d'une résolution haute auront plus d'importance dans le nombre d'attributs du descripteur HOG final. A cause de ce déséquilibre, la basse résolution a finalement moins d'importance que la haute résolution. Cependant, dans beaucoup de cas d'étude, notamment la classification d'objets urbains, cela ne convient pas au problème. Prenons l'exemple d'un objet circulaire (comme une bouche d'égout) qui est très facilement classable en utilisant une résolution basse. En effet, les différentes orientations des contours se compensent pour former un histogramme HOG équilibré. A l'inverse, à haute résolution, la description du bord se fait en plusieurs positions ce qui la rend plus sensible aux bruits et donc plus difficilement interprétable (cf figure 6.2).

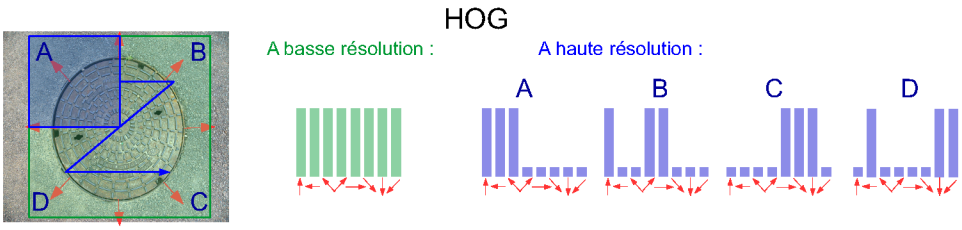


FIGURE 6.2 – Représentation des histogrammes HOG pour deux résolutions différentes. En vert, la basse résolution considère l'ensemble de l'imagette. En bleu, la haute résolution ne considère qu'une partie de l'imagette en divisant la fenêtre en quatre zones.

Dans [90] nous mettons également en avant ce problème. Si nous quantifions $\mathbf{f}^r$ en mots visuels, la représentation pour chaque résolution a alors la même taille et dépend uniquement de la taille du dictionnaire utilisé. Ainsi la quantification et l'uniformisation des fréquences de mots visuels à différentes résolutions augmentent les performances d'environ 9% par rapport à une approche plus classique [123]. Cependant la quantification en mots visuels, dans le traitement d'images de grandes dimensions, est trop coûteuse en temps de calculs et fut donc mise de coté.

Afin d'éviter le problème de déséquilibre lors des fusions des résolutions, nous proposons d'utiliser un nouveau type de réseaux de SVM. Celui-ci est introduit dans la prochaine section.

6.2 Proposition d'un réseau de SVM

6.2.1 Définition d'un réseau de SVM

Un réseau de SVM est un réseau se composant d'un ensemble de SVM qui s'agencent de façon analogue à celle des neurones dans un réseau de neurones. Un réseau de SVM est une succession de couches se composant de plusieurs SVM. Nous considérons une couche d'entrée, des couches cachées et une couche de sortie. Contrairement à un réseau de neurones, dont les poids se déterminent par minimisation globale de l'erreur, les SVM au sein du réseau de SVM effectuent leur apprentissage les uns après les autres par maximisation de marges (voir la section 3.1). L'apprentissage s'effectue couche par couche, de la couche d'entrée vers la couche de sortie. En effet, chaque SVM de la première couche effectue son apprentissage de manière indépendante. Puis l'opération est répétée sur la couche suivante qui prend en entrée les sorties de la couche précédente. Contrairement aux neurones classiques, où les poids sont initialisés aléatoirement selon une distribution, les SVM ne considèrent pas de phénomènes aléatoires durant leurs apprentissages. Ainsi, si deux SVM prennent en paramètre les même caractéristiques, leur apprentissage convergera vers le même modèle. Afin d'obtenir au sein d'une même couche de SVM des modèles variés, il est donc nécessaire qu'ils reçoivent en paramètre différentes caractéristiques. Pour cela, chaque SVM ne reçoit en paramètre qu'un sous-ensemble des sorties de la couche précédente. Le réseau ainsi construit n'est donc pas entièrement connecté comme la plupart des réseaux neuronaux.

Les SVM appartenant aux couches cachées sont aléatoirement connectés à la sortie de certains SVM des couches précédentes. Le nombre de connexions est défini aléatoirement dans un intervalle $\{C_{min}, .., C_{max}\}$ fixé par l'utilisateur. Nous notons $n_i^{(c)}$ le nombre de connexions du $i^{ème}$ SVM de la couche c avec $n_i^{(c)} \in \{C_{min}, C_{max}\}$. Le nombre de SVM sur la couche c est noté $N^{(c)}$. La couche de sortie ne comporte qu'un seul SVM qui est connecté à tous les SVM de la couche précédente.

A l'intérieur du réseau, les SVM utilisés peuvent comporter des fonctions noyaux de type gaussien, polynomial, linéaire, etc... Dans le cas linéaire, le score retourné par un SVM correspond au produit scalaire entre son poids et le vecteur d'entrée, noté $s_i^{(c)}$ avec i le numéro du SVM sur la couche c. Le réseau effectue alors une combinaison linéaire de scores et l'utilisation d'un réseau n'a alors plus d'intérêt. De plus, cela ne nous permet pas de normaliser les différentes sorties des SVM entre elles et n'élimine pas des signaux saturés. Afin d'éviter ce problème nous projetons les sorties des SVMs sur une distribution sigmoïde asymétrique, voir équation 6.1.

$$p_i^{(c)} = \frac{1}{1 + e^{-\alpha_i^{(c)} s_i^{(c)} + \lambda_i^{(c)}}} \tag{6.1}$$

avec $p_i^{(c)}$ la sortie du SVM après la non linéarité, $\alpha_i^{(c)}$ et $\lambda_i^{(c)}$ des paramètres propres à chaque SVM qui sont déterminés après l'apprentissage de chaque SVM. Pour déterminer les meilleures valeurs, l'algorithme de Platt est utilisé conformément à [41]. Cette méthode se base sur l'estimation du maximum de vraisemblances qui est calculé sur une base de validation. Dans le cas binaire, pour un SVM fixé nous cherchons à calculer les meilleurs paramètres α et λ. Nous notons $p_i \in [0..1]$ la sortie du SVM définie par l'équation 6.1 appliquée au $i^{ème}$ élément de la base de validation avec $i \in \{0, .., N_{val}\}$ et $y_i \in \{0, 1\}$ le label correspondant. Dans un premier temps, l'algorithme de Platt transforme le label afin de prendre en considération l'asymétrie de la base de données. Dans notre cas, il y a davantage d'objets n'étant pas des tombes (y=0) que des objets étant des tombes (y=1). Ainsi, nous notons t_i l'étiquette pondérée telle que :

$$t_i = \begin{array}{ll} \frac{N_{val}^{autre}+1}{N_{val}^{autre}+2} & si\ y_i = 0 \\ \frac{1}{N_{val}^{tombe}+2} & si\ y_i = 1 \end{array}$$

avec N_{val}^{tombe} et N_{val}^{autre} le nombre d'objets portant le label tombe et ne portant pas le label tombe dans la base de validation. Dans un second temps, pour trouver les meilleurs paramètres α et λ, l'algorithme de Platt minimise l'équation suivante :

$$(\alpha, \lambda) = \underset{(\alpha,\lambda)}{argmax} \sum_{i=1}^{N_{val}} (t_i log(p_i) + (1 - t_i) log(1 - p_i))$$

Les nœuds de SVM possèdent donc de façon analogue aux neurones une fonction d'activation. Dans la suite, nous utiliserons le terme neurone pour désigner un nœud de SVM dans le réseau de SVM. Le SVM présent sur la couche de sortie retourne la probabilité d'appartenance aux différentes classes.

L'architecture des couches d'entrées dépend des caractéristiques passées en paramètres. Dans la section suivante nous détaillerons comment créer un réseau de SVM avec des descripteurs HOG.

6.2.2 Utiliser les descripteurs HOG avec un réseau de SVM

La première couche d'un réseau de SVM est primordiale car elle représente la façon avec laquelle l'on traite l'information. Nous avons mis en évidence dans la partie précédente le problème de fusion des résolutions au sein du descripteur HOG. Chaque résolution du descripteur HOG est transmis à un seul SVM d'entrée évitant ainsi les problèmes de fusion de résolutions. Chaque SVM

effectue son apprentissage sur une seule résolution. Les décisions résultantes pour chaque résolution sont alors fusionnées dans les couches cachées (cf schéma 6.3).

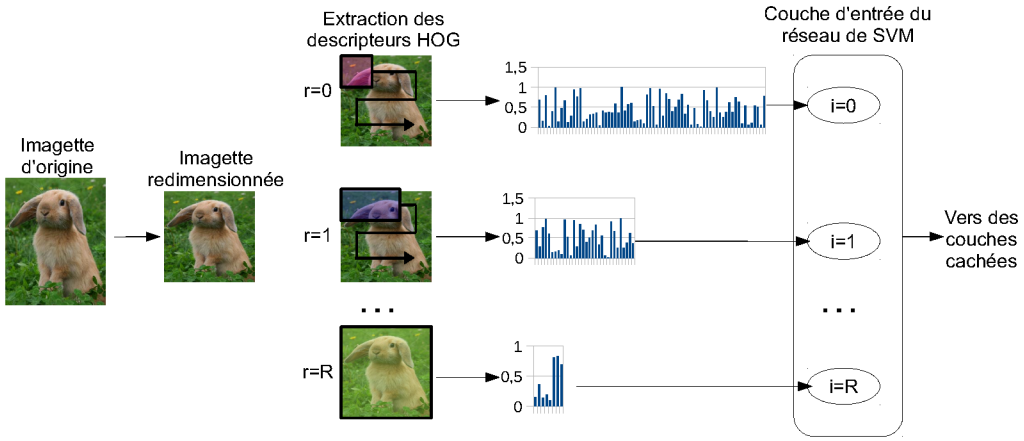


FIGURE 6.3 – Illustration du passage de descripteur HOG à un réseau de SVM.

Dans cette architecture, chaque SVM de la couche d'entrée ne prend en paramètre qu'une seule résolution. Cependant, certaines combinaisons de résolutions pourraient mener à une meilleure classification globale. Nous souhaitons donc ajouter des neurones d'entrés qui seraient connectés à différentes résolutions du descripteur HOG. Une solution est d'utiliser un algorithme type Adaboost [39, 40] afin de trouver les meilleurs sous-ensembles de caractéristiques. Cependant cette méthode ne permet pas de trouver des ensembles performants en considérant les sorties des autres SVM du réseau. Afin de trouver des combinaisons pouvant enrichir l'information à l'intérieur du réseau nous définissons un nouveau type de neurone : *le neurone aléatoire* (*random neuron*). Celui-ci appartient à la couche d'entrée et est aléatoirement connecté à des sous-ensembles du descripteur HOG. Il permet l'introduction de variations au sein de réseaux en mélangeant à la fois les résolutions et les positions d'extraction des fenêtres glissantes dans le HOG.

Dans la sous section suivante nous présenterons les résultats obtenus avec cette architecture qui a été acceptée lors du workshop IEEE Machine Learning for Signal Processing [85].

6.2.3 Définir les paramètres du réseau de SVM

Plusieurs paramètres restent à définir lors de la création d'un réseau de SVM à savoir :

— le nombre de neurones aléatoires,
— le nombre de couches,
— la taille de chaque couche.

Pour déterminer ces paramètres, nous avons évalué leurs impacts sur une base de validation. Effectuer une recherche optimale globale serait très coûteux en temps d'apprentissage nous avons donc opté pour une autre stratégie qui consiste à fixer les paramètres les uns à la suite des autres. Dans un premier temps, nous figeons la taille du réseau à une seule couche contenant 400 neurones cachés. Puis, nous faisons varier le nombre de neurones aléatoires pour trouver celui donnant les meilleures performances. Pour évaluer les performances des réseaux de SVM, nous calculons la précision moyenne pour un rappel se trouvant dans l'intervalle allant de 45 à 80%. Dans le cas de la détection de tombes, la figure 6.4.A permet de trouver le nombre optimal de SVM aléatoires. Nous constatons que la courbe semble avoir une asymptote vers 74% de précision pour un nombre de SVM aléatoires supérieur ou égal à 200. Afin de minimiser les calculs, nous gardons le nombre minimal de SVM requis pour obtenir les meilleures performances soient 200 neurones aléatoires dans notre cas.

Après avoir figé la nombre de neurones aléatoires, nous cherchons le meilleur nombre de SVM dans la première couche cachée. Pour cela nous fixons $C_{min} = 40$ et $C_{max} = 100$. Sur la figure 6.4.B, nous pouvons voir un pic de performances avec une précision de 74.1% pour 300 neurones cachés. Si nous ajoutons plus de neurones cachés, le réseau sur-apprend la base d'apprentissage et les performances chutent rapidement. Nous répétons le processus en rajoutant des couches cachées jusqu'à ce que les performances n'augmentent plus. Au final, nous obtenons un réseau avec deux couches cachées contenant chacune 300 SVM.

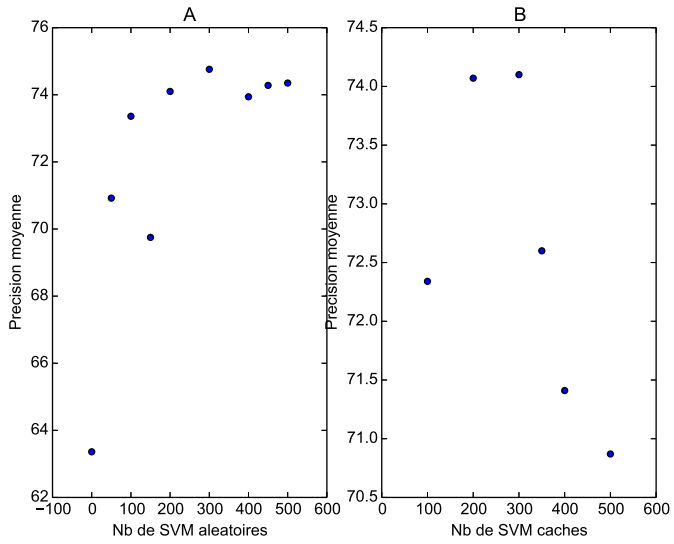


FIGURE 6.4 – Courbes de précision moyenne calculées sur la base de validation pour un rappel allant de 45 à 80%. La sous figure A permet d'évaluer l'importance des SVM aléatoires. Le sous figure B permet d'évaluer l'importance de la taille du réseau.

6.2.4 Premiers résultats d'un réseau de SVM

La figure 6.5 compare les performances d'un réseau de SVM utilisant les paramètres définis précédemment et un unique SVM utilisant la totalité du descripteur HOG. Nous constatons que pour un rappel élevé, la précision est d'environ 40% pour les deux méthodes. Aucun gain n'est donc obtenu par notre approche. En effet, certaines tombes ne sont jamais détectées par aucune des deux méthodes. Le plus souvent la représentation de ces tombes ne comporte pas de dalle, n'est pas alignée avec les autres et possède une forte végétation. La figure 6.6 représente une image de la base de tests. On peut y voir des tombes très peu visibles se confondant avec le fond. Pour la plupart des objets considérés comme difficilement détectables, nous interprétons rapidement qu'il s'agit de tombes grâce à l'ombre portée. Cependant, cette ombre est rarement présente et est donc peu apprise lors de la classification. Nous souhaitons proposer une approche fonctionnant avec ou sans la présence d'ombres. Dans ce contexte, il est clair que les objets encadrés en orange de la figure 6.6 seront souvent confondus avec le sol.

Pour comparer les méthodes, nous étudions les performances pour un rappel compris entre 40 et 80%. Dans cet intervalle, les réseaux de SVM sont beaucoup plus performants que l'utilisation d'un SVM simple avec un gain moyen de 15% en précision. Par exemple, nous notons que pour un rappel de 70% le gain en précision est alors de 19%.

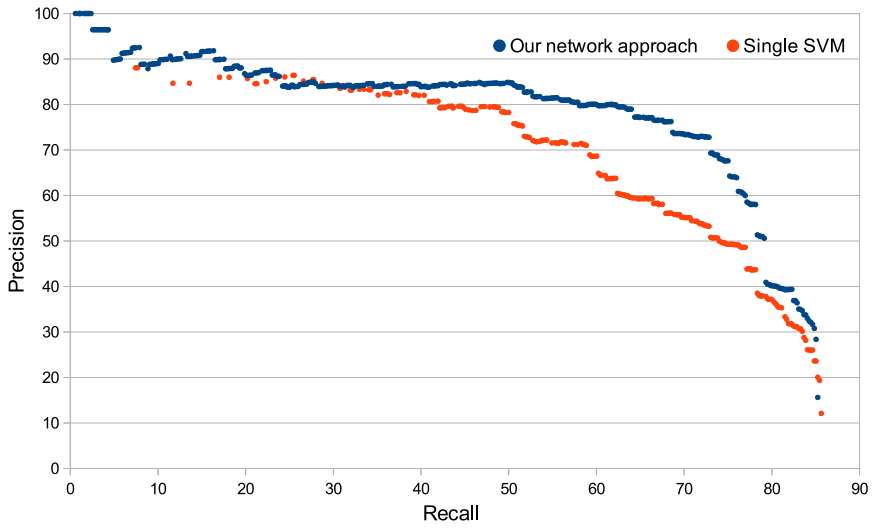


FIGURE 6.5 – Évaluation du réseau de SVM (bleu) comparé à un seul SVM (orange) sur la détection de tombes.

Dans le contexte de la détection et de la localisation de tombes, les réseaux de SVM permettent d'obtenir un gain élevé. Dans cette section, nous avons uniquement utilisé les descripteurs HOG calculés sur des images en niveaux de gris. Cependant, nous possédons aussi l'information couleur qu'il serait intéressant d'utiliser pour mieux détecter les tombes, plus particulièrement

FIGURE 6.6 – Exemples d'objets difficilement détectables avec le descripteur HOG.

pour les cas difficiles illustrés sur la figure 6.6. Dans la section suivante, nous présenterons la façon dont le réseau utilise l'information couleur et la création d'une architecture permettant une meilleure exploitation de celle-ci.

6.3 Exploitation de la couleur par un réseau de SVM

6.3.1 Étude préliminaire sur la couleur et le réseau

6.3.1.1 Quelques informations autour de la couleur

Dans notre cas d'étude, la couleur semble moins pertinente que les formes présentes sur les images. Cependant, elle peut être très complémentaire et apporter l'information nécessaire pour mieux reconnaître certains types d'objets. La figure 6.7 montre l'importance de la couleur pour les zones ombragées.

Le plus souvent l'information couleur est représentée par un triplet d'entiers dont chaque composante représente la couleur rouge, verte et bleue (RGB). Cette information peut se représenter au travers d'autres espaces possédant diverses propriétés.

Nous pouvons par exemple disposer d'espaces avec une seule composante représentant des niveaux de gris. Nous définissons deux espaces à une seule dimension. La première consiste à effectuer la moyenne des trois composantes RGB. La seconde consiste à calculer la luminance, notée Y définit par : $Y = 0.21R + 0.71G + 0.07B$ [4].

Dans l'état de l'art il existe de nombreux espaces couleur composés de trois canaux couleurs. Dans cette thèse, nous avons utilisé et comparé certains d'entre eux, à savoir :

— CIE-LAB et CIE-LUV [83] inspirés de la représentation des couleurs chez

FIGURE 6.7 – Différence entre une image couleur et une image en niveau de gris. L'image en niveau de gris est calculée à partir de la moyenne des composantes couleur.

l'humain. Il s'agit de systèmes reposant sur une représentation des couleurs en $CIE-XYZ$. Le système XYZ est un système à trois composantes où la luminance Y est séparée des autres composantes, à savoir, les chrominances : X et Z. La chrominance représente la colorimétrie d'une couleur alors que la luminance représente son intensité. Le passage d'un système RGB vers un système XYZ se fait suivant l'équation :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41 & 0.35 & 0.18 \\ 0.21 & 0.71 & 0.07 \\ 0.01 & 0.11 & 0.95 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{6.2}$$

Les coefficients de $CIE-LAB$ sont alors définis comme étant :

$$\begin{aligned} L &= 116\, f(\tfrac{Y}{Y_n}) - 16 \\ a &= 500 \left(f(\tfrac{X}{X_n}) - f(\tfrac{Y}{Y_n}) \right) \\ b &= 200 \left(f(\tfrac{Y}{Y_n}) - f(\tfrac{Z}{Z_n}) \right) \end{aligned} \quad \text{avec } f(x) = \begin{cases} x^{0.33} & si\ x > \left(\tfrac{29}{6}\right)^2 \\ 0.33 \left(\tfrac{29}{6}\right)^2 x + \tfrac{4}{29} & \text{sinon} \end{cases} \tag{6.3}$$

avec X_n, Y_n et Z_n les valeurs de X, Y, Z pour une couleur blanche d'étalonnage. L est appelée la clarté et représente une forme dérivée de la luminance.

— HSV [57] est une représentation de la teinte, de la saturation et de l'intensité (Hue, Saturation and Value HSV), elle est définie comme :

$$H = \begin{cases} 0 & si\ max = min \\ \left(60\frac{g-b}{max-min} + 360\right)\ \%360 & max = r \\ 60\frac{b-r}{max-min} + 120 & max = g \\ 60\frac{r-g}{max-min} + 240 & max = b \end{cases}$$

$$S = \begin{cases} 0 & si\ max = 0 \\ 1 - \frac{min}{max} & \text{sinon} \end{cases}$$

$$V = max$$

avec max et min la valeur maximale et la valeur minimale entre les composantes RGB.

De nombreuses méthodes et de nouveaux espaces couleur permettent d'obtenir une bonne définition de la couleur [53, 27]. Ces espaces couleur donnent de meilleurs taux de classification selon les contextes. Dans la suite, nous utiliserons que les espaces les plus connus.

6.3.1.2 Utilisation directe de la couleur

Le descripteur HOG s'extrait sur une seule composante et ne permet donc pas d'utiliser les trois composantes couleur. Certains auteurs ont proposé des extensions aux descripteurs de formes en n'utilisant qu'une seule composante ou un mélange de composantes [1, 81]. Cependant en réalisant cette opération, de l'information portée par une composante peut-être perdue. Une solution est d'extraire le descripteur HOG sur chaque composante couleur et de les concaténer dans un seul vecteur de caractéristiques. Cependant, cette solution est coûteuse en mémoire et en temps d'apprentissage puisqu'elle multiplie par trois la taille des descripteurs pour les SVM ayant une complexité non linéaire.

Nous pouvons utiliser le réseau de SVM de la même façon que pour les HOG multi-résolutions : les différents descripteurs extraits depuis différentes composantes couleur sont donnés à différents neurones d'entrée. Nous nommons cette architecture de réseaux, l'architecture du réseau de *SVM séparée*, car l'information est divisée sur les neurones de la couche d'entrée (Cf figure 6.8).

6.3.1.3 Comparaison des différents espaces couleur

Nous allons maintenant comparer l'utilisation d'un réseau de SVM séparé avec un unique SVM prenant en paramètre la concaténation des descripteurs HOG issus des différentes composantes couleur dans différents espaces couleur. L'architecture du réseau de SVM est déterminée à l'aide d'une base de validation comme dans la section 6.2.3. Le réseau de SVM séparé est ainsi constitué de 400 neurones aléatoires sur la première couche et de deux couches contenant 600 neurones cachés chacune.

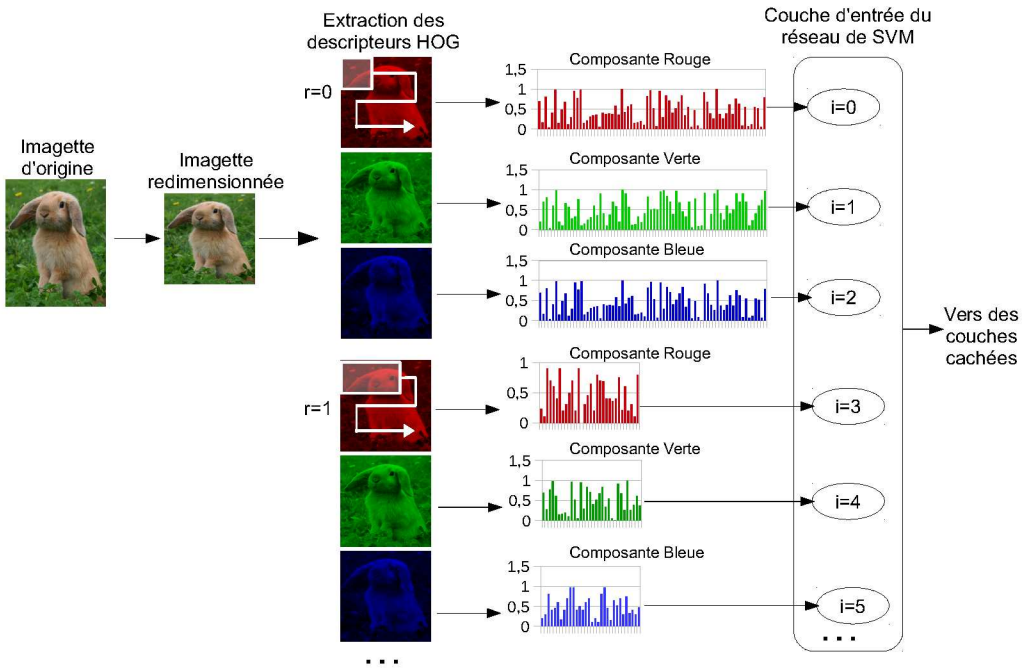


FIGURE 6.8 – Représentation du traitement de la couleur dans l'architecture séparée.

La figure 6.9 permet de visualiser et de comparer les résultats pour le réseau de SVM et pour le SVM unique en fonction de l'espace couleur choisi. Dans un premier temps, nous pouvons comparer les espaces ne possédant qu'une seule composante, à savoir Y et L. La clarté L permet de gagner en performance par rapport à la luminance. Le gain en précision étant de l'ordre de 4% quel que soit le rappel. Dans le cas d'un SVM unique, la composante de clarté est plus performante que l'utilisation d'un espace couleur à trois composantes. Par exemple, les performances pour un rappel de 60% est de 78% de précision pour la clarté contre 75% dans le cas des composantes RGB. Ce résultat est paradoxal car même s'il y a plus d'informations grâce à la couleur, il met en évidence le problème de fusion d'informations provenant de différentes sources. L'information rajoutée n'augmente pas les performances de classification et provoque même une perte de précision.

Dans un second temps, nous nous intéressons aux résultats provenant du réseau de SVM séparé. La première conclusion évidente est que le réseau de SVM séparé a toujours de meilleures performances que l'unique SVM, confirmant ainsi les performances du chapitre précédent sur les images en niveaux de gris. Le graphique 6.9 met en évidence que les performances du réseau n'utilisant que la clarté sont plus efficaces que celles utilisant les espaces LAB, LUV et même HSV. Cela peut être provoquer par deux phénomènes. Le premier est un phénomène de sur-apprentissage sur des images ayant un type de couleur particulier dans ces espaces. Le second montre que le réseau gère mal les informations inutiles. En effet, les connexions des couches cachées étant

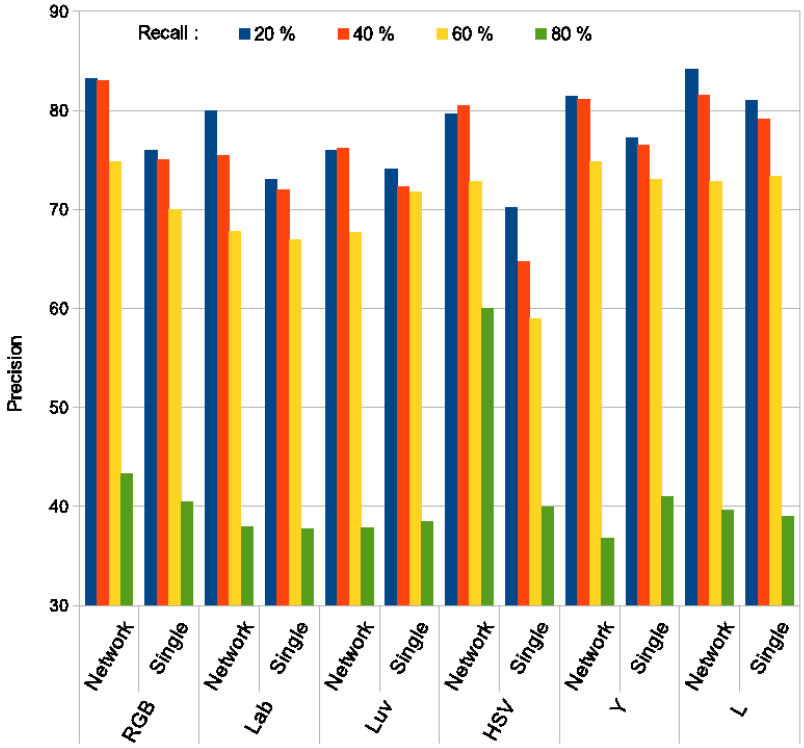


FIGURE 6.9 – Représentation en bâtons des performances par le biais de la précision moyenne calculée pour une valeur de rappel. Les couleurs bleue, orange, jaune et verte sont utilisées pour la représentation des précisions qui sont calculées pour un rappel de 25, 40, 60 et 80%.

aléatoires, il est possible que certains neurones cachés n'utilisent que l'information provenant de la chrominance, qui dans notre cas est moins discriminante, provoquant ainsi une baisse de performances. Pour éviter ce problème, il serait intéressant d'éliminer les neurones cachés possédant une précision faible sur une base de validation. Notons que pour l'espace HSV, même si les performances moyennes sont inférieures, dans le cas où nous cherchons à détecter le plus de tombes possible (rappel élevé), la précision est très largement supérieure à celle des autres approches. Pour un rappel de 80%, la précision obtenue est alors de 60% contre 44% pour l'espace RGB ou 38% pour la clarté. Ce gain est causé par certaines tombes difficilement détectables qui sont partiellement recouvertes d'ombres. Or, l'espace HSV, par sa représentation de la couleur est peu affecté par ce type phénomène. En effet, les ombres vont provoquer une grande variation d'intensité mais modifieront peu la teinte et la saturation.

Finalement, contrairement à l'utilisation d'un unique SVM les meilleures performances pour un réseau de SVM séparé sont directement issues de l'utilisation des composantes RGB. Un gain moyen en précision de 3% est alors possible par rapport à une architecture basée uniquement sur la clarté. Ce gain reste cependant relativement faible puisque le réseau ne traite pas les différentes composantes couleur de manière fine. Dans la section suivante, nous montre-

rons que le réseau peut, avec une architecture différente, mieux considérer l'information couleur et augmenter les performances.

6.3.2 Architecture avancée pour la couleur

6.3.2.1 Présentation de l'architecture par fusion et par agrégation

Afin de mieux considérer l'information couleur, nous proposons plusieurs architectures modifiant la couche d'entrée du réseau de SVM séparé. L'idée est de mieux considérer l'information couleur, telle que l'information RGB que nous prendrons en exemple dans cette section. Dans l'architecture séparée, les descripteurs HOG provenant des composantes couleurs et des résolutions sont séparés et transmis à différents neurones. Or, l'information de la résolution peut être conservée et elle peut apporter de la représentativité. Nous appelons *architecture par fusion*, l'architecture permettant de regrouper et concaténer les différents descripteurs HOG d'une même résolution. Le nombre de SVM sur la couche d'entrée est donc équivalent à celui obtenu par le traitement d'une image en niveaux de gris (Cf figure 6.10). Lors de cette opération, un problème émerge : les différents canaux sont fusionnés et il peut donc y avoir des problèmes de normalisation. Dans le cas de la représentation couleur *RGB* cela a peu importance, mais notons que cette architecture peut être difficile à étendre pour fusionner des descripteurs de différentes natures.

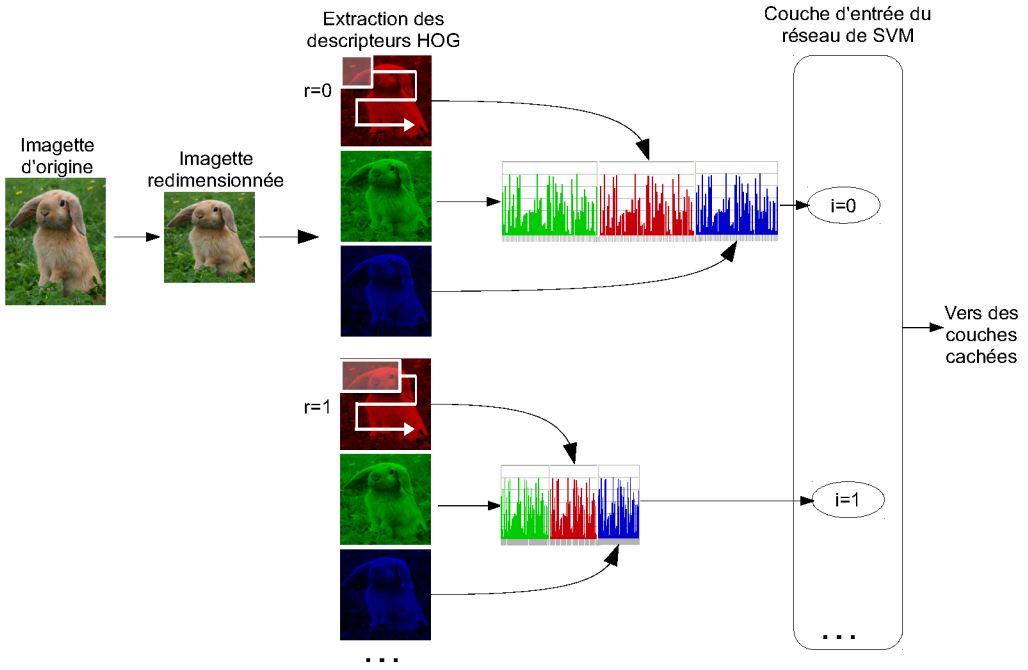


FIGURE 6.10 – Illustration d'un réseau de SVM avec l'architecture par fusion.

Nous souhaitons mettre en place une architecture générique capable de traiter la fusion de différents types de caractéristiques, car l'architecture par fusion n'est pas en adéquation avec cette théorie. Par contre, nous pouvons rajouter

une couche particulière connectée à la couche d'entrée de l'architecture séparée. Cette couche serait capable d'agréger et de fusionner les résultats pour une même résolution. Ainsi l'information est correctement traitée, de façon séparée, puis est fusionnée au sein de la seconde couche. Nous avons implémenté et comparé différents types d'opérations. La seconde couche peut fusionner les résultats en effectuant :

— une opération linéaire qui retourne le produit, on parle alors *d'architecture par produit*,
— une opération non linéaire qui retourne le maximum, on parle alors *d'architecture par maximum*.
— une opération linéaire qui retourne la première composante d'une analyse en composante principale (ACP).

La figure 6.11 met en avant la surcouche de ce type de réseaux par rapport à une architecture séparée.

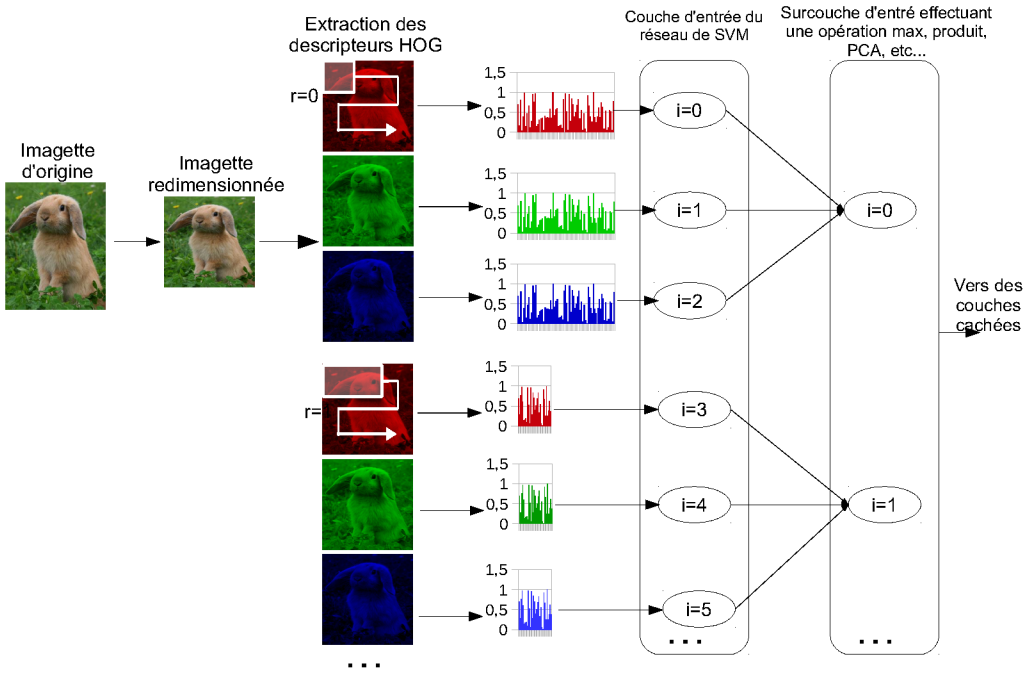


FIGURE 6.11 – Illustration d'un réseau de SVM avec l'architecture par opération de type maximum, produit, PCA...

L'avantage de l'architecture par opération est que les différentes informations, dans notre cas les canaux couleur, sont traitées séparément. L'agrégation pour une résolution donnée transforme les signaux, issus des vecteurs HOG calculés sur le triplet de couleurs, en un réel. Cette transformation provoque une perte non négligeable de l'information. Néanmoins, l'architecture par fusion permet d'obtenir un traitement simultané des différentes informations pour une résolution donnée. Notons tout de même que cette architecture ne peut pas fusionner des informations de natures différentes sans se confronter à un problème important de normalisation.

6.3.2.2 Présentation de l'architecture empilée

Nous proposons donc un nouveau type d'architecture permettant de normaliser l'information séparément et de l'analyser de façon groupée. Nous appelons cette architecture : *l'architecture empilée*. L'idée consiste à utiliser une architecture séparée pour traiter les différents canaux. Les scores retournés par les SVM servent à pondérer les valeurs des descripteurs HOG d'entrée. L'ensemble des vecteurs HOG pour une résolution donnée est ensuite concaténé et permet l'apprentissage d'un SVM sur une seconde couche (cf figure 6.12). Par exemple, dans la représentation RGB, nous définissons $\{p_0^0, p_3^0, p_6^0, ..., p_{3R}^0\}$,

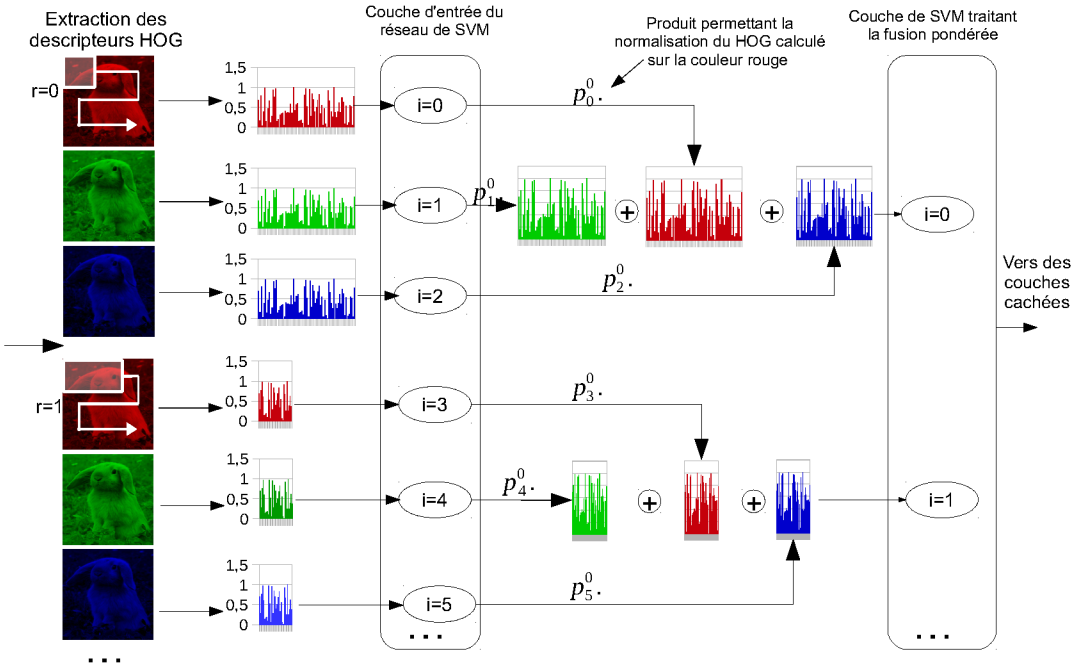


FIGURE 6.12 – Illustration d'un réseau de SVM avec l'architecture empilement.

$\{p_1^0, p_4^0, p_7^0, ..., p_{3R+1}^0\}$ et $\{p_2^0, p_5^0, p_8^0, ..., p_{3R+2}^0\}$ comme étant les probabilités retournées par les SVM de la couche d'entrée traitant des HOG extraits sur les composantes rouges, vertes et bleues. Prenons $f_r^{(c)}$ le descripteur HOG extrait sur un canal couleur $c \in \{R, G, B\}$ pour une résolution r. Le vecteur d'entrée noté f'_r, transmis au réseau de SVM après pondération pour une résolution r est définie dans l'équation 6.4. Cette architecture proposée permet donc de normaliser automatiquement et intelligemment les informations de différentes natures en entrée. Notons que la complexité est importante car cette architecture possède de nombreuse neurones traitant des vecteurs de grandes tailles (la concaténation des descripteurs en entrée).

$$f'_r = p_r^0 \times f_r^R \oplus p_{r+1}^0 \times f_r^G \oplus p_{r+2}^0 \times f_r^B \tag{6.4}$$

avec $\oplus$ l'opérateur de concaténation des vecteurs.

6.3.2.3 Évaluation des différentes architectures

Chacune des architectures a été évaluée dans les mêmes conditions en utilisant les composantes couleurs RGB en entrée. Sur le graphique 6.13 nous évaluons et représentons les courbes ROC. Nous constatons que l'architecture

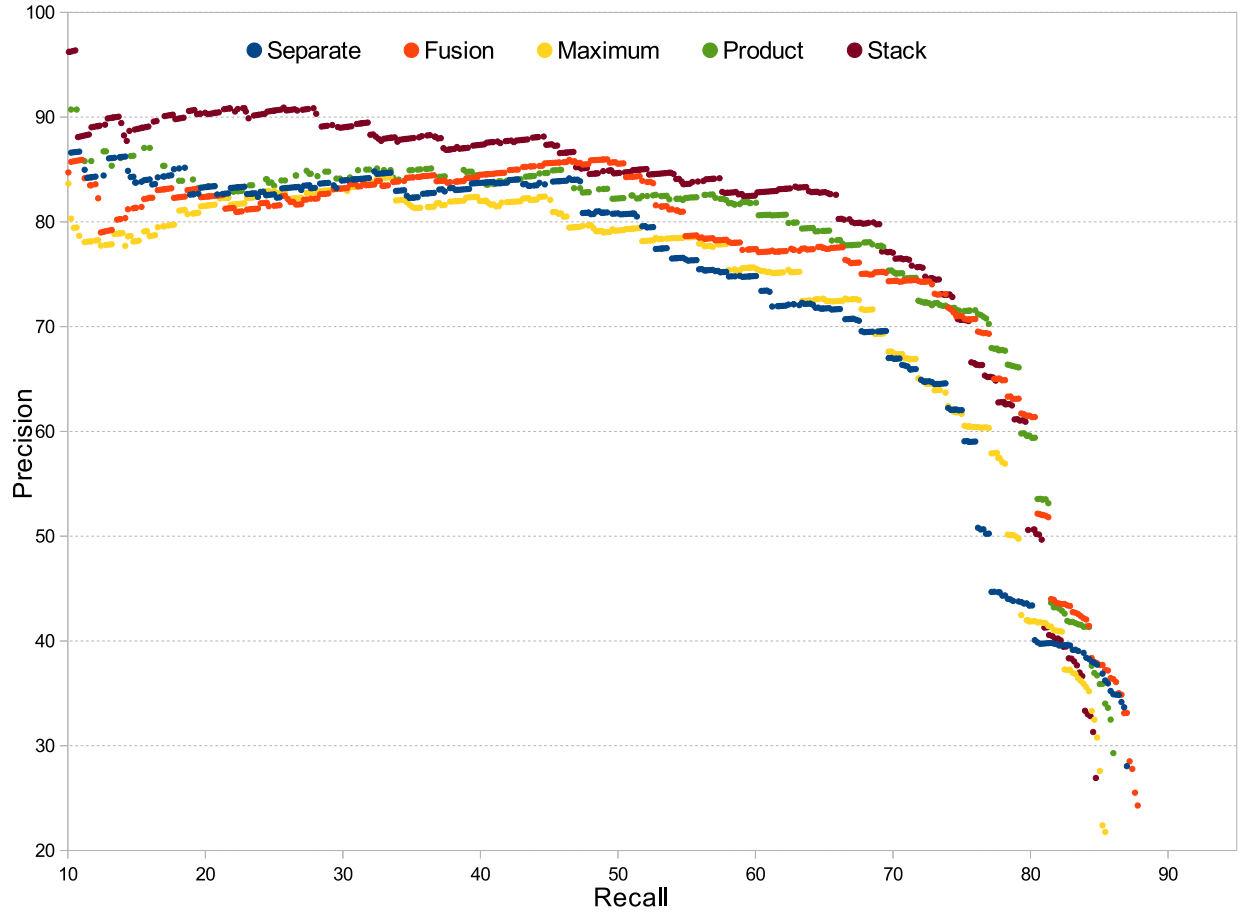


FIGURE 6.13 – Courbe ROC des performances comparant les différentes architectures d'un réseau de SVM.

par séparation est la moins performante, avec par exemple pour un rappel de 60% une précision de 75% contre plus de 83% pour les architectures de produit et d'empilement. L'information au sein du réseau est probablement mal analysée et la nécessité de diminuer l'information en amont des couches cachées est importante. Avec les mêmes performances, l'architecture par maximum n'est également pas fonctionnelle. Quantifier le signal en ne gardant que le score avec la plus forte valeur n'est pas une solution viable comparée aux autres architectures. Nous pouvons alors supposer que l'information apportée par des couleurs "secondaires" est importante et qu'il est nécessaire de la conserver. L'architecture par fusion présente de bonnes performances pour un rappel supérieur à 75%. Cependant, pour un rappel de 60%, elle reste 5% moins efficace qu'une approche par produit. De plus, nous avons vu que l'architecture par fusion ne fonctionne par pour les descripteurs de différentes natures.

Les deux architectures par produit et par empilement sont les plus performantes avec une précision d'environ 83% pour un rappel de 60%. Cependant pour tout rappel inférieur à 50%, la précision de l'architecture par empilement est supérieure en moyenne à 6%. De plus, pour un rappel entre 50 et 75% la précision moyenne de l'architecture par empilement est accrue de 5%. Lorsque le rappel est supérieur à 75%, l'architecture par produit devient plus performante d'environ 3% mais les performances chutent rapidement vers des résultats inexploitables. L'architecture par empilement est donc en moyenne bien plus performante. De plus, toute l'information, pour une résolution donnée, est traitée simultanément par un seul SVM en étant normalisée par la prédiction de SVM en amont.

Cette architecture pourrait permettre une meilleure fusion de différents types de caractéristiques comme les descripteurs HOG avec des descripteurs LBP mais des tests et expérimentations sont nécessaire pour le confirmer.

Nous avons vu que les réseaux de SVM permettent un bien meilleur traitement de la couleur que l'utilisation d'un unique SVM. De plus, en étant correctement structurés, l'information est optimisée et les performances sont bien meilleures [89]. L'utilisation d'un réseau avec les canaux RGB permet une augmentation d'environ 10% par rapport à un SVM unique sur son meilleur espace couleur, à savoir la clarté. Cependant rajouter et activer tous ces SVM au sein d'un réseau a un coût calculatoire non négligeable. Dans la section suivante, nous proposerons une méthode pour réduire le coût en calculs d'activation d'un réseau de SVM.

Chapitre 7

Réduction des temps d'évaluation avec des chemins d'activation

7.1 Introduction au problème calculatoire

Un des problèmes majeurs de l'approche objet est le coût calculatoire lors de la phase d'évaluation. En effet, la fenêtre glissante évalue toutes les positions et résolutions d'une image de tests, ce qui, pour une image de dimension 2000×2000 pixels représente plusieurs millions d'évaluations. Il est donc essentiel que chacune des évaluations soit le plus rapide possible. Nous avons vu qu'utiliser une cascade de classifieurs [123, 116] permet de réduire considérablement le coût de cette phase. Cependant, ces méthodes montrent plusieurs faiblesses. La première réside dans la structure même de la cascade. L'utilisateur doit définir de nombreux paramètres comme le nombre d'étages dans la cascade et les conditions de sorties requises sur chacun d'entre eux. Ces paramètres empiriques ne garantissent pas d'obtenir les meilleures performances comme le montre [73]. Par ailleurs, pour obtenir des réductions importantes de coût, il est nécessaire d'avoir une cascade possédant de nombreux étages. Or, chaque étage augmente la probabilité d'avoir une erreur, ce qui implique que les performances des cascades sont inférieures à celles d'un classifieur unique [33, 73]. Ce phénomène est d'autant plus marqué pour les objets peu représentés dans la base d'apprentissage ou très variables [51].

Dans la section précédente, nous avons détaillé le fonctionnement d'un réseau de SVM. Mais, il n'est pas intéressant d'effectuer directement des cascades de réseaux de SVM car cela provoquerait une perte de performances. Par contre, nous pouvons adapter le concept de cascades afin qu'il soit optimisé pour des réseaux de SVM. Cette amélioration s'effectue par ce que l'on nommera le *chemin d'activation* qui est introduit dans la section suivante.

7.2 Définition du chemin d'activation

7.2.1 Fonctionnement d'un réseau d'activation

Nous avons vu que, lors de la phase dite d'activation, le réseau de SVM active l'ensemble de ses classifieurs les uns à la suite des autres. Cependant, dans un réseau de SVM, plusieurs SVM peuvent s'activer en même temps. Il est alors nécessaire de les prioriser et de les ordonnancer. L'ordre d'activation des SVM forme ce que nous appellerons le *chemin d'activation*.

Lorsque le chemin d'activation est entièrement parcouru l'ensemble des SVM est activé ce qui implique un coût calculatoire important. L'idée du chemin d'activation est de définir un ordre de telle façon à ce que le coût calculatoire soit réduit. Pour cela, lors de l'activation, chaque SVM activé peut stopper la traversée du chemin et donc l'activation du réseau. Pour stopper l'activation du réseau, lors de l'activation, chaque SVM va émettre une probabilité représentant l'appartenance à une classe. Lorsque cette probabilité est importante, le SVM est "sûr" de sa décision et celle-ci est probablement correcte. L'idée du chemin d'activation est alors de stopper l'exploration du réseau à ce niveau et d'émettre la probabilité trouvée du SVM "sûr".

Dans le cas d'une classification binaire, nous notons (a) et (b) les deux classes. Le chemin d'activation doit définir deux seuils, notés $\theta_i^{(a),(c)}$ et $\theta_i^{(b),(c)}$, représentant les seuils de rejet d'un SVM numéro i sur la couche c. Le seuil $\theta_i^{(x),(c)}$ représente la valeur minimale à atteindre pour que l'objet activant le SVM soit considéré de la classe x. Une base de validation est utilisée pour déterminer automatiquement la valeur de ces seuils. Pour cela, nous considérons que chaque SVM doit avoir une précision minimale sur cette base. Nous noterons $p_{min}(x)_i^{(c)}$ la précision minimale pour la classe $x \in \{a, b\}$. Si nous notons $Rappel_i^{(c)}(\theta)$ et $Precision_i^{(c)}(\theta)$ les deux fonctions retournant le rappel et la précision du SVM i sur la couche c en considérant un seuil θ, nous obtenons l'équation :

$$\theta_i^{(x),(c)} = \underset{\theta \in [0..1]}{argmax}\ Rappel_i^{(c)}(\theta)\ /\ Precision_i^{(c)}(\theta) \geq p_{min}(x)_i^{(c)} \tag{7.1}$$

Dans nos expériences, nous utiliserons la même précision minimale pour régler les valeurs $\theta_i^{(s),(c)}$ et $\theta_i^{(t),(c)}$. Nous posons alors $p_{min_i}{}^{(c)} = p_{min}(a)_i^{(c)} = p_{min}(b)_i^{(c)}$.

Nous définissons le coût d'activation d'un neurone SVM comme le nombre de produits que le réseau va devoir effectuer pour activer le SVM. Ce coût comprend le coût d'activation des parents qui est non négligeable. Sur l'exemple de la figure 7.1 le coût d'activation du neurone sur la couche 0 numéro 0 est de 3. En effet, il effectue uniquement un produit scalaire entre un vecteur d'entrée de dimension 3 et son poids. Par contre, le neurone sur la couche 1 numéro 0

possède un vecteur d'entrée de taille 2 et a un coût de 7. Cette valeur comptabilise le coût de ces deux parents qui sont respectivement de 3 et 2 ainsi que son propre coût de produit scalaire qui est de 2.

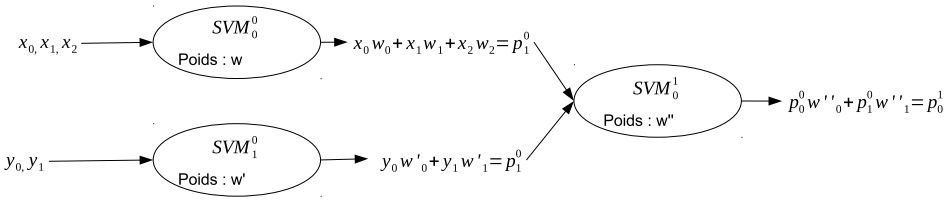


FIGURE 7.1 – Exemple montrant les opérations effectuées lors de l'activation d'un réseau de SVM.

Une fois les paramètres de seuils trouvés, chaque SVM peut être caractérisé par trois attributs : son rappel et sa précision sur une base de validation et son coût d'activation.

Il est possible que certains SVM ne possèdent aucune solution respectant la condition sur la précision minimale. On appellera ces SVM des *SVM inertes*. Les SVM inertes ne font pas directement partie du chemin d'activation car ils ne possèdent pas de règles de rejets. Cependant, ils peuvent jouer un rôle dans le coût d'activation de SVM qui leurs sont connectés. Ces trois composantes vont permettre de construire un chemin comme il est décrit dans la sous-section suivante.

7.2.2 Ordonnancement du chemin d'activation

Afin d'ordonner les SVM dans le chemin d'activation nous proposons trois solutions.
La première solution consiste à construire le chemin d'activation de façon à activer en premier les classifieurs peu coûteux.
La seconde solution consiste à activer en premier les classifieurs avec une forte probabilité de stopper l'activation et donc de gagner en coût de calculs. Cependant, il s'agit le plus souvent de SVM dans les couches cachées, nécessitant préalablement l'activation d'autres SVM qui eux sont potentiellement peu importants.
La troisième solution est d'effectuer un compromis entre le coût d'activation et la probabilité de stopper l'exploration afin d'obtenir un score de performance du SVM. Ce score est alors utilisé pour ordonnancer les SVM dans le chemin. Cependant, effectuer un compromis est difficile car cela nécessite de fusionner la valeur de coût d'activation, la précision et le rappel.

Nous considérons que le problème de construction du chemin est équivalent à un problème d'exploration de graphe. A la première itération, seuls les

éléments de la couche d'entrée sont accessibles, puis avec l'activation de ces éléments ceux des couches supérieures le deviennent. Sur la figure 7.2, on peut voir deux exemples de méthodes de construction de chemin d'activation.
La première, nommée par performance, effectue une construction de façon à stopper l'exploration du graphe le plus vite possible et va donc chercher les SVM accessibles avec des rappels élevés.
Le second, appelé par coût, active toujours le SVM le moins coûteux et en cas d'égalité, le SVM le plus performant sera utilisé.
On constate que la stratégie consistant à activer les SVM les moins coûteux en premier permet d'aller plus rapidement en profondeur dans le graphe et d'activer plus vite des SVM de la couche 1. Nous souhaitons encourager l'activation du réseau en profondeur car les performances y sont souvent meilleures. Pour cela, la construction des chemins s'effectue par la méthode cherchant à réduire le coût.

Une fois défini, le chemin va permettre d'activer les SVM dans un ordre fixé. De plus, chaque SVM non-inerte possède deux seuils, $\theta_i^{(a),(c)}$ et $\theta_i^{(b),(c)}$, permettant de stopper l'exploration s'il est suffisamment sûr de sa décision. Cependant, il est possible que les SVM se trompent et plus le réseau est grand, plus les chances de commettre une erreur augmentent. Afin de réduire l'erreur, nous proposons de rendre les conditions d'arrêts plus strictes dans la section suivante.

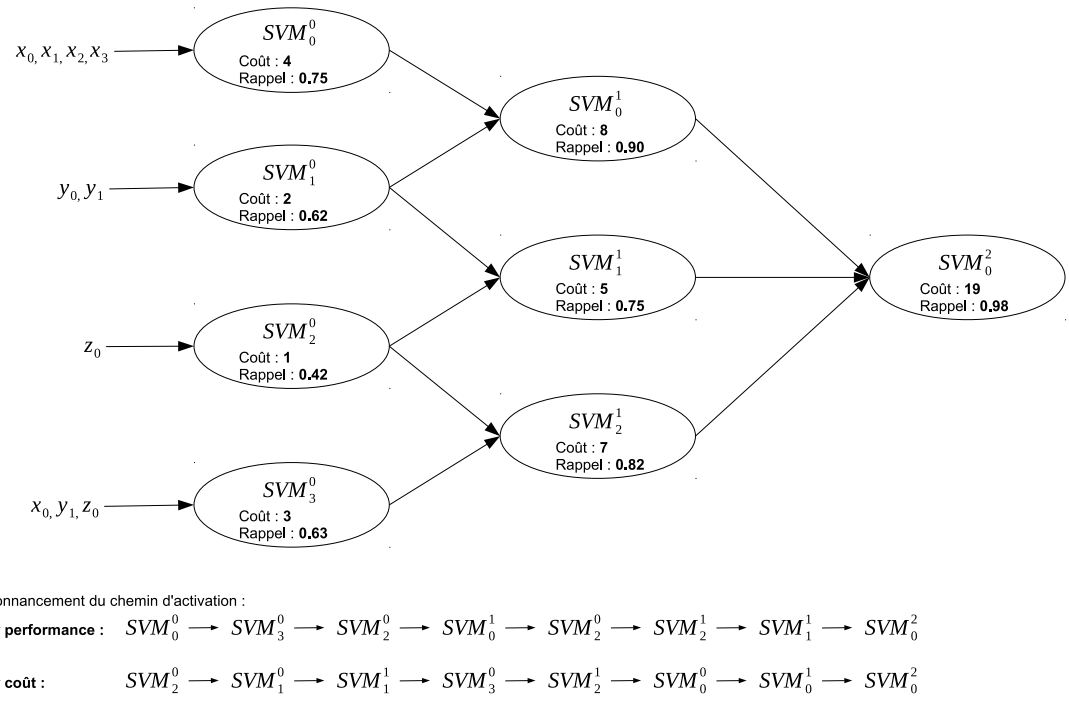


FIGURE 7.2 – Exemple de construction de l'ordonnance de deux chemins d'activation selon le coût calculatoire et selon les performances des SVMs.

7.2.3 Critère d'arrêt et confiance

Dès que le score d'un SVM est supérieur à un des seuils $\theta_i^{(a),(c)}$ et $\theta_i^{(b),(c)}$ l'activation du réseau s'arrête. Cependant, cette décision peut-être erronée en fonction de la probabilité ${p_{min_i}}^{(c)}$ utilisée lors de l'apprentissage. Dans le cas d'un réseau de grande dimension, de manière analogue au fonctionnement interne d'une cascade, l'erreur finale est accrue. Pour lutter contre ce problème nous proposons de ne pas stopper l'activation lorsqu'un SVM est sûr de sa décision. Nous définissons alors un ensemble de SVM, appelé *SVM de confiance*, qui a un score supérieur à leurs seuils $\theta_i^{(a),(c)}$ et $\theta_i^{(b),(c)}$ et arrête donc l'activation du réseau.

Formellement nous fusionnons les décisions des SVM de confiances et notons P la somme des décisions. Plus P est petit et négatif plus l'objet activant le réseau va tendre à être de la classe a, à l'inverse, plus P est un grand nombre positif plus l'objet tend vers la classe b. Dans le cas binaire, $p_i^{(c)}$ représente la probabilité que l'objet soit de la classe a et donc $1 - p_i^{(c)}$ la probabilité qu'il soit de la classe b. Lors de l'activation du réseau chaque SVM numéro i appartenant à une couche quelconque c modifie P s'il est de confiance, suivant l'équation :

$$\begin{cases} p_i^{(c)} > \theta_i^{(a),(c)} & \text{alors } P = P - 1 \\ p_i^{(c)} < 1 - \theta_i^{(b),(c)} & \text{alors } P = P + 1 \\ \theta_i^{(a),(c)} \geqslant p_i^{(c)} \geqslant 1 - \theta_i^{(b),(c)} & SVM_i^{(c)} \text{ pas confiant} \end{cases} \tag{7.2}$$

P est donc une variable permettant d'accumuler les preuves que l'objet est d'une certaine classe. Lorsque P dépasse le *seuil de confiance minimal*, noté θ_{lim}, fixé par l'utilisateur, l'activation du réseau est stoppée. L'équation 7.3 formalise la condition d'arrêt du parcours du chemin d'activation :

$$\begin{cases} P < -\theta_{lim} & \rightarrow \text{L'objet est de la classe } a \\ P > \theta_{lim} & \rightarrow \text{L'objet est de la classe } b \\ \text{sinon} & \text{on continue l'activation du réseau} \end{cases} \tag{7.3}$$

Plus le seuil de confiance minimal est grand, plus la précision du réseau sera importante. En effet, plusieurs SVM seront interrogés et voteront majoritairement pour la classe prédite ce qui réduit la probabilité d'erreurs. Si nous considérons que $\theta_{lim} = 1$, la probabilité p_f d'obtenir un faux positif est : $p_f = 1 - \prod p_{min_i}^{(c)}$ pour tous les SVMs compris dans le chemin d'activation. Si $p_{min_i}^{(c)}$ est constant quel que soit la couche et le numéro de SVM nous obtenons alors : $p_f = 1 - (p_{min})^n$ avec n le nombre de SVM dans le chemin.
Cependant, si nous augmentons la valeur du paramètre θ_{lim} la condition d'arrêt est plus difficile à atteindre et les SVMs commettent moins d'erreurs. Nous avons cherché à modéliser la probabilité de faux positifs en fonction du pa-

ramètre θ_{lim}. Il est important de considérer que tous les SVM apprennent sur une base identique, ce qui implique que les votes des SVM ne sont pas décorrélés. Ainsi, lorsqu'un SVM se trompe, les suivants peuvent répéter la même erreur. Dans le cas, où chaque SVM apprend sur une base différente de celle des autres, la probabilité de faux positifs peut s'exprimer comme l'indique l'équation 7.4. Dans notre cas, cela représente un gain maximal atteignable avec l'introduction d'un seuil de confiance minimal.

$$p_f = 1 - (p_{min_i}^{(c)})^{\frac{n}{\theta_{lim}}} \tag{7.4}$$

Cependant lorsque le seuil de confiance est élevé, le temps nécessaire avant de couper l'activation augmente et la réduction de temps de calculs est moins significative. Il est donc nécessaire d'ajuster ce paramètre pour obtenir de bonnes performances sans augmenter le coût calculatoire.

7.3 Évaluation expérimentale

Dans cette section nous évaluons les performances du chemin d'activation dans les mêmes conditions et avec le même réseau que celui utilisé dans la section 6.2.4. Il s'agit d'un réseau apprenant sur des vecteurs HOG multi-résolutions avec deux couches cachées de 300 neurones chacune. Nous posons $\theta_{lim} = 1$ ce qui implique que le premier SVM confiant stoppe l'activation et que sa probabilité est utilisée pour la fenêtre englobante. L'objectif est de réduire les temps en calculs afin de permettre un traitement, à l'aide d'une fenêtre glissante, quasiment temps réel, c'est-à-dire l'évaluation d'une image de dimension 5000×5000 pixels en moins de 5 minutes.

Dans cette première évaluation, nous disposons de deux classes à séparer à savoir la classe *tombe* et la classe *non tombe*. Nous considérons uniquement des coupures d'exploration à l'aide du chemin d'activation que si l'objet trouvé est une tombe, c'est à dire $p_i^{(c)} > \theta_i^{(tombe),(c)}$. Aussi nous ne chercherons pas à éliminer le plus vite possible les nombreux faux positifs. Cette supposition peu courante est due au fait que les images étudiées sont en majeure partie composées de tombes et donc il est plus intéressant de détecter rapidement les objets de type tombe que les objets n'en étant pas.

Nous proposons deux manières de construire le chemin d'activation. La première consiste à faire varier le p_{min} de façon à ce que les premiers SVM activés rejettent moins facilement les fenêtres. Ainsi, p_{min} est initialisé à une valeur élevée qui va décroître de manière linéaire en fonction de la position du SVM sur le chemin d'activation. En effet, nous supposons que les SVM sont moins fiables au début du réseau que sur les couches plus profondes, car elle possèdent moins d'informations. Pour cela, nous faisons varier p_{min} de

manière linéaire dans le chemin d'activation de la valeur 0.99 à la valeur 0.75. La deuxième solution consiste à figer p_{min} à une constante identique pour tous les SVMs. Ainsi, lors des premiers tests, nous figeons p_{min} à 0.95.

Approches	Temps hh :mm	Précision moyenne
SVM simple	15 :54	60.06%
Réseau de SVM	24 :35	75.0%
Chemin d'activation p_{min} décroissant	4 :58	75.3%
Chemin d'activation p_{min} constant	5 :48	74.5%

TABLE 7.1 – Comparaison des moyennes de temps d'exécution et de la précision pour les différentes approches. La précision est calculée sur une plage de rappel allant de 20 à 80%.

Les résultats donnés sur le tableau 7.1 évaluent notre base de tests avec la machine du HPC@LR [1]. Lors de l'apprentissage les calculs sont parallélisés sur 12 cœurs et leur durée n'est pas évaluée. Lors de la phase de tests, aucun n'est parallélisée (pour éviter des défauts d'implémentation) et les évaluations sont réalisées avec un seul cœur Intel Xeon 2.66GHz.

En interprétant le tableau 7.1, nous constatons que le temps d'évaluation d'un réseau de SVM est environ 150% plus long en temps de calculs que l'utilisation d'un SVM unique. Dans ces conditions, bien que les performances soient meilleures de 15%, il est difficile d'envisager une application effectuant un changement d'échelle avec l'utilisation de milliers d'images. Le chemin d'activation avec un p_{min} décroissant permet une réduction des temps de calculs du réseau passant de plus de 24 h à moins de 5 h. La diminution en temps de calculs est telle, que celui-ci devient plus rapide en phase d'évaluation qu'un SVM unique. De plus, nous constatons une légère amélioration des performances moyennes. En effet, les SVM peu performants sont détectés comme étant des SVM inertes et ne permettent pas de stopper l'activation afin de ne pas réduire les performances. De plus, certains SVM se "spécialisent" à l'intérieur du réseau et possédent alors un rappel très faible mais une précision très élevée. Ce type de SVM ne permet pas réellement de réduire le temps de calculs, car les coupures dans l'activation sont très rares, mais ils permettent de bien détecter des tombes qui sont considérées comme du sol par les autres SVM.

Le chemin d'activation avec un p_{min} constant apparaît moins performant que le chemin avec un p_{min} décroissant. Cependant, le seuil choisi n'est pas optimal car il a été fixé arbitrairement. Pour correctement évaluer cette méthode, il est nécessaire d'effectuer des tests avec une validation croisée pour trouver le meilleur seuillage sur la base de validation. Le résultat de cette recherche est donné dans sur le tableau 7.2. A l'aide d'une base de validation, nous avons

1. Centre de Compétences en calcul haute performance de la région Languedoc-Roussillon HPC@LR.

effectué une recherche du meilleur paramètre p_{min} de classification de façon à toujours obtenir une précision moyenne supérieure à 75% (cf tableau 7.2).
Pour un $p_{min} > 0.92$ les temps d'évaluation semblent converger. Afin de garantir une bonne précision, nous gardons la valeur de p_{min} la plus élevée possédant la meilleure réduction calculatoire, c'est à dire $p_{min} = 0.92$. Sur la base de tests, en considérant cette valeur de p_{min}, le temps d'évaluation est réduit de plus de 30 min par rapport à l'approche avec un p_{min} croissant, soit un gain relatif de 11% en temps de calculs. De plus, les performances augmentent de 1% par rapport au réseau de SVM classique.

p_{min}	Temps base de validation (%)	Temps base de tests (HH :MM)	Précision moyenne base de tests
0.97	100%	11 :02	75.3
0.95	62.1%	5 :48	74.5
0.92	**61.0%**	**4 :25**	**76.0**
0.90	61.2%	4 :22	75.23
0.88	59.9%	4 :32	75.3
0.85	60.1%	4 :15	74.6
0.80	61.0%	4 :12	71.5
0.99→0.75		**4 :58**	**75.3**

TABLE 7.2 – Tableau évaluant les performances du réseau en fonction de la valeur de p_{min}. La première colonne représente le coût calculatoire sur la base de validation. Les deuxième et troisième colonnes représentent le coût calculatoire et la précision moyenne sur la base de tests. La dernière ligne rappelle les performances du chemin d'activation avec une valeur de p_{min} décroissante.

Dans cette section nous avons étudié la première facette des chemins d'activation dans le cas d'un réseau de SVM se basant sur des descripteurs HOG multi-résolutions. Nous montrons que le chemin d'activation réduit le temps d'évaluation de plus de 20h, passant d'une évaluation de 24h35 à 4h25 soit un gain relatif de 82%. Par ailleurs, le chemin d'activation permet, par la mise en avant de SVM de confiance, d'augmenter la précision moyenne de 1%. Cette technique permet l'utilisation du réseau de SVM pour une application réelle le rendant même plus performant.
Cependant, les résultats présentés reposent sur l'utilisation du descripteur HOG. Dans la prochaine section, nous allons nous affranchir de ce descripteur et nous évaluerons dans un premier temps les méthodes d'apprentissage profond sur nos données. Puis, dans un second temps, nous adapterons les chemins d'activation à ces méthodes d'apprentissage profond.

Chapitre 8

Intégration des CNN au réseau de SVM

8.1 Introduction au CNN utilisé

Les CNN sont des méthodes d'apprentissage objet, effectuant leurs apprentissages sur des imagettes et utilisant une fenêtre glissante lors de la phase d'évaluation, voir la section 5.3. Dans notre cas, comme indiqué dans la section 5.3, la taille de ces imagettes est de 64×64 pixels. Mais, les réseaux classiquement utilisés effectuent le plus souvent la classification d'images de tailles 256 par 256 pixels. Il faut donc les adapter afin de pouvoir utiliser des structures similaires. Dans notre étude, nous avons conservé le réseau AlexNet [65] composé de 5 couches de convolutions, 3 couches de sous-échantillonnage et 2 couches entièrement connectées. Cependant, nous n'utilisons plus de chevauchement de 4 pixels sur la première couche de convolutions et comme le montre la figure 8.1, nous avons décalé la seconde couche de sous-échantillonnage derrière la troisième couche de convolutions afin de perdre le moins d'informations possible. En effet, les images étant de plus petites tailles, les chevauchements initiaux ne sont plus nécessaires pour réduire la quantité d'information à traiter. De la même façon, les sous-échantillonnages peuvent être effectués plus en profondeur.

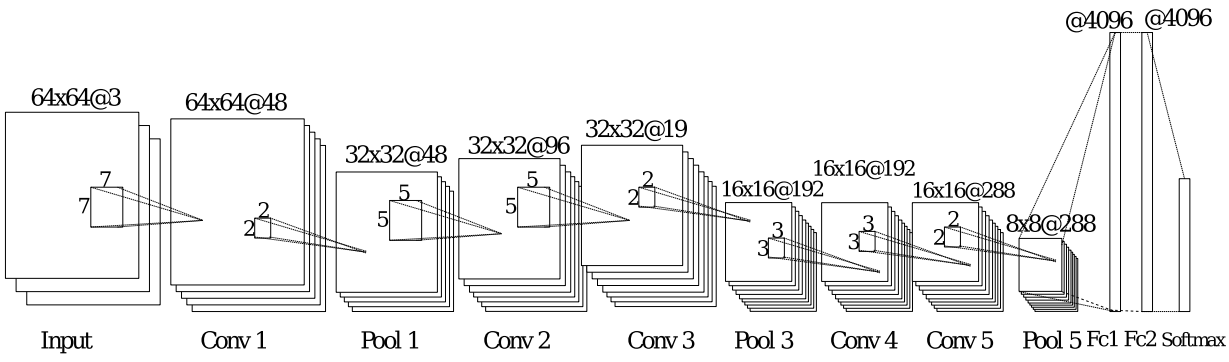


FIGURE 8.1 – Architecture du réseau de neurones convolutifs utilisée.

Pendant l'apprentissage d'un CNN il est classique d'augmenter le volume

de données en entrée du réseau. Pour cela, chaque imagette est tournée et inversée selon les axes verticaux et horizontaux pour obtenir 6 images transformées. Par ailleurs, lors de l'extraction d'un objet à une position donnée, on extrait également les imagettes avec un décalage vertical et horizontal allant jusqu'à 4 pixel par pas de 2, ce qui permet d'obtenir 24 images transformées. Chaque imagette va subir les transformations indiquées ou un mélange de celles-ci, amplifiant le nombre de tombes par un facteur 100. Le nombre total de tombes en entrée du réseau dans la base d'apprentissage augmente jusqu'à plus de 1 340 000.

8.1.1 Premiers résultats du CNN

Nous avons d'abord testé le CNN seul et il s'avère qu'un nombre très important de faux positifs est trouvé, puisque la précision est de 12% (cf tableau 8.1). Comme nous le montre la figure 8.2, en plus des erreurs de détection, il semble que le réseau centre mal les fenêtres englobantes.
Nous supposons que le problème de recalage est causé par la fusion des fenêtres englobantes qui ont toutes une probabilité égale à 99%. Ainsi, lors de la fusion des résultats, le post-traitement conserve une fenêtre aléatoire ce qui provoque de nombreuses erreurs. Le premier problème est donc la mauvaise répartition des scores de probabilités.
De plus, le CNN détecte de nombreux faux positifs comme les toits, les murs et la route. Ces objets font partie de la catégorie 'autre' dans la base d'apprentissage mais ne sont pas assez représentés pour permettre une bonne reconnaissance.

Toutes ces erreurs sont évitables et dans la sous-section suivante nous proposons de modifier la base d'apprentissage pour s'en affranchir.

8.1.2 Amplification de la base d'apprentissage

Nous constatons que le CNN possède de très nombreux faux positifs. La base de vrai négative étant généré aléatoirement, elle ne permet pas de représenter la diversité des objets n'étant pas des tombes. Une solution serait de construire cette base de vrai négatif itérativement de façon à ré injecter dans l'apprentissage des objets faux positifs détectés [42]. Dans cette thèse, afin de réduire l'erreur nous allons effectué ce processus de manière empirique. En effet, après un premier apprentissage sur une base de données, nous constatons que les faux positifs sont principalement situés sur les toits et murs. Afin de réduire le nombre d'erreurs commises par le CNN, nous avons donc créé des sous-classes pour ces objets. Pour cela, nous avons re-effectué des étiquetages et avons rajouté les classes : 'toit' et 'mur'. Chacune de ces classes contient des parties de tailles variables de toit et de mur. De plus, nous avons créé la classe

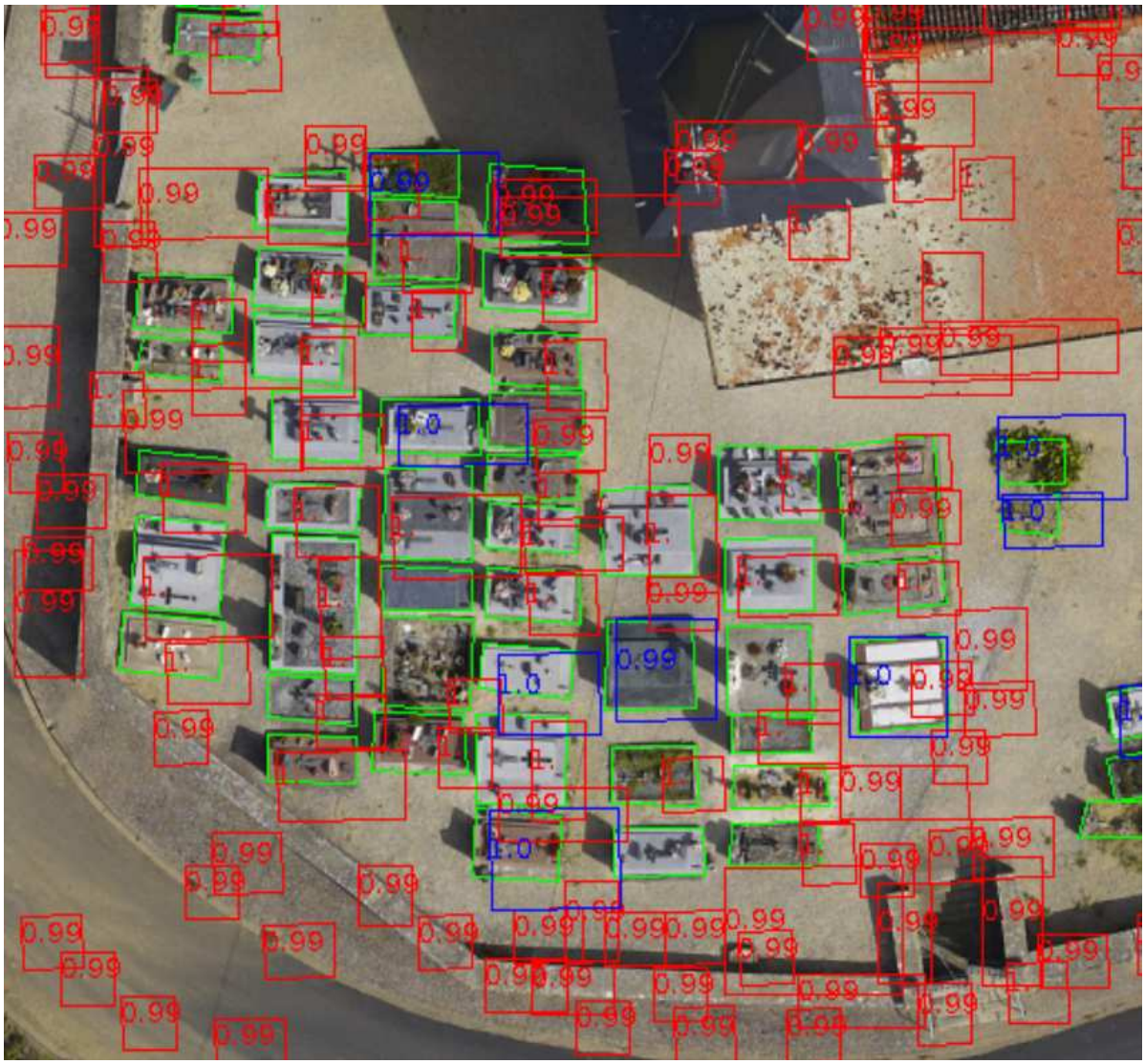


FIGURE 8.2 – Résultats de la détection par CNN sur une image de tests, les cadres représentés en vert sont la vérité terrain, en bleu les vrais positifs et en rouge les faux positifs. Les fenêtres englobantes possèdent un score qui correspond à la probabilité de contenir une tombe.

'semi-tombe' contenant la moitié d'une tombe. La classe semi-tombe effectue une translation de 50% horizontalement et verticalement pour extraire des imagettes autour de la tombe considérée (cf figure 8.3).

Le volume des données d'entrées a augmenté considérablement comme le montre le tableau 8.1. Sur celui-ci, nous voyons l'importance de donner différentes classes à reconnaître pour le CNN. En effet, plus le CNN va apprendre à reconnaître les classes ressemblants aux tombes, plus il va être capable de discriminer les objets d'intérêt de ceux qui y ressemblent. Ainsi, le gain est de l'ordre de 32% en précision entre un scénario ne considérant que les objets recherchés (scénario I) et celui considérant quatre types d'objets (scénario IV). Cependant, il faut alors noter que la taille de la base est augmentée d'environ 20%, ce qui rallonge les temps d'apprentissage du même ordre de grandeur.

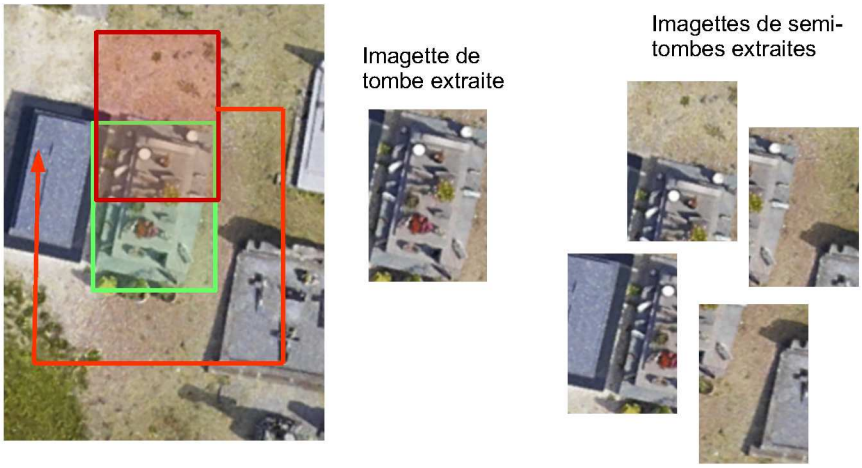


FIGURE 8.3 – Exemple de semi-tombes extraites autour de la tombe encadrée en vert.

Scenarii	I	II	III	IV
Nb tombes	1.34×10^6	1.34×10^6	1.34×10^6	1.34×10^6
NB autres	1.6×10^6	1.6×10^6	$1.6x \times 10^6$	1.6×10^6
Nb semi-tombes		1.34×10^6	1.34×10^6	1.34×10^6
Nb toits			4.03×10^5	4.03×10^5
Nb murs				7×10^5
Précision	12%	23%	34%	45%

TABLE 8.1 – Représentation de la précision pour la classe tombe sur la base de validation, pour un rappel fixé à 70%, pour différents CNN apprenant sur une base d'apprentissage de plus en plus complète.

8.2 Combiner un CNN et un réseau de SVM

Nous avons vu dans le chapitre 6 que les réseaux de SVM permettent un traitement adapté des caractéristiques multi-résolutions. Un réseau de neurones convolutifs permet d'extraire des caractéristiques optimisées pour un problème donné sur ces différentes couches. Or, plus la couche est profonde, plus elle a été convoluée et sous-échantillonnée. La profondeur de la couche d'extraction définit donc une résolution de caractéristiques. Nous souhaitons appliquer les avantages des réseaux de SVM pour le traitement de la résolution dans le cas de caractéristiques extraites à partir d'un CNN. Notre objectif est de mieux traiter les différentes caractéristiques du CNN et d'utiliser le concept du chemin d'activation (cf chapitre 7) afin de réduire le temps de calculs lors de la phase d'évaluation. Nous allons donc interconnecter au sein d'un même réseau des neurones classiques du CNN et des neurones SVM. Les neurones s'organiseront en couches, chacune notée $L^{(c)}$ avec c le nom de la couche.

Dans le cas du CNN, nous reprenons le réseau de la section 8.1. Nous dis-

posons donc d'un réseau de 11 couches tel que :

$$c \in \{Conv1,\ Pool1,\ Conv2,\ Conv3,\ Pool2,\ Conv4,\ Conv5,\ Pool5,\ FC1,\ FC2,\ Softmax\}$$

Chaque couche du réseau CNN correspond à un niveau d'abstraction : plus la couche est profonde, plus le réseau apprend des concepts complexes. Une fois l'apprentissage du CNN effectué les caractéristiques retournées par les différentes couches sont très intéressantes car elles sont adaptées au problème de classification. Nous allons maintenant connecter le réseau de SVM aux différentes couches du CNN et donc à différents niveaux d'abstraction.

Le réseau de SVM reçoit en entrée les cartes de caractéristiques de CNN. Pour cela, le réseau de SVM est constitué de plusieurs couches d'entrées. Chaque couche d'entrée ne prend en paramètre qu'une seule couche du réseau CNN.

Les couches d'entrées du réseau de SVM sont constituées de deux types de neurones SVM (cf figure 8.4) :

— Les neurones SVM standards qui prennent en paramètre la totalité d'une et une seule carte de caractéristiques. Dans chaque couche d'entrée de SVM il y a autant de SVM standards que de cartes de caractéristiques.

— Les neurones SVM à multiples cartes qui prennent en paramètre un nombre fixé de cartes de caractéristiques. De manière analogue aux SVM aléatoires, décrits dans 6.2.1, les cartes de caractéristiques utilisées sont sélectionnées aléatoirement dans une couche donnée.

Une fois les entrées du réseau de SVM créées, le réseau s'agence d'une succession de couches cachées constituées de SVM possédant un nombre de connexions aléatoires. Pour des raisons calculatoires nous ne pouvons pas connecter une couche de SVM après chacune des couches du CNN. Il est alors nécessaire de choisir des points d'intérêt stratégiques. Dans le cas du réseau proposé dans la figure 8.1, nous proposons de connecter le réseau de SVM après chaque sortie des couches de sous-échantillonnages. Ce choix se justifie car, à ces positions, l'information portée par la carte de caractéristiques est sous-échantillonnée (dans notre cas divisée par 4) ce qui rend le traitement plus rapide. En plus de ces trois points d'ancrage, nous connectons deux SVM après les couches entièrement connectées du CNN. En effet, le coût de ces couches est négligeable comme elles sont constituées d'uniquement 4096 neurones retournant des scalaires.

La figure 8.5 permet de visualiser l'architecture du réseau de SVM connecté au CNN de la figure 8.1. Nous proposons d'utiliser 9 couches de SVM tel que : L^{SVM1} est connectée à L^{Pool1}, L^{SVM3} est connectée à L^{Pool3}, L^{SVM5} est connectée à L^{Pool5}, L^{SVM7} est connectée à L^{FC1}, L^{SVM8} est connectée à L^{FC2}, L^{SVM2} prend en entrée la couche L^{SVM1}, L^{SVM4} prend en entrée la couche L^{SVM2} et L^{SVM3},

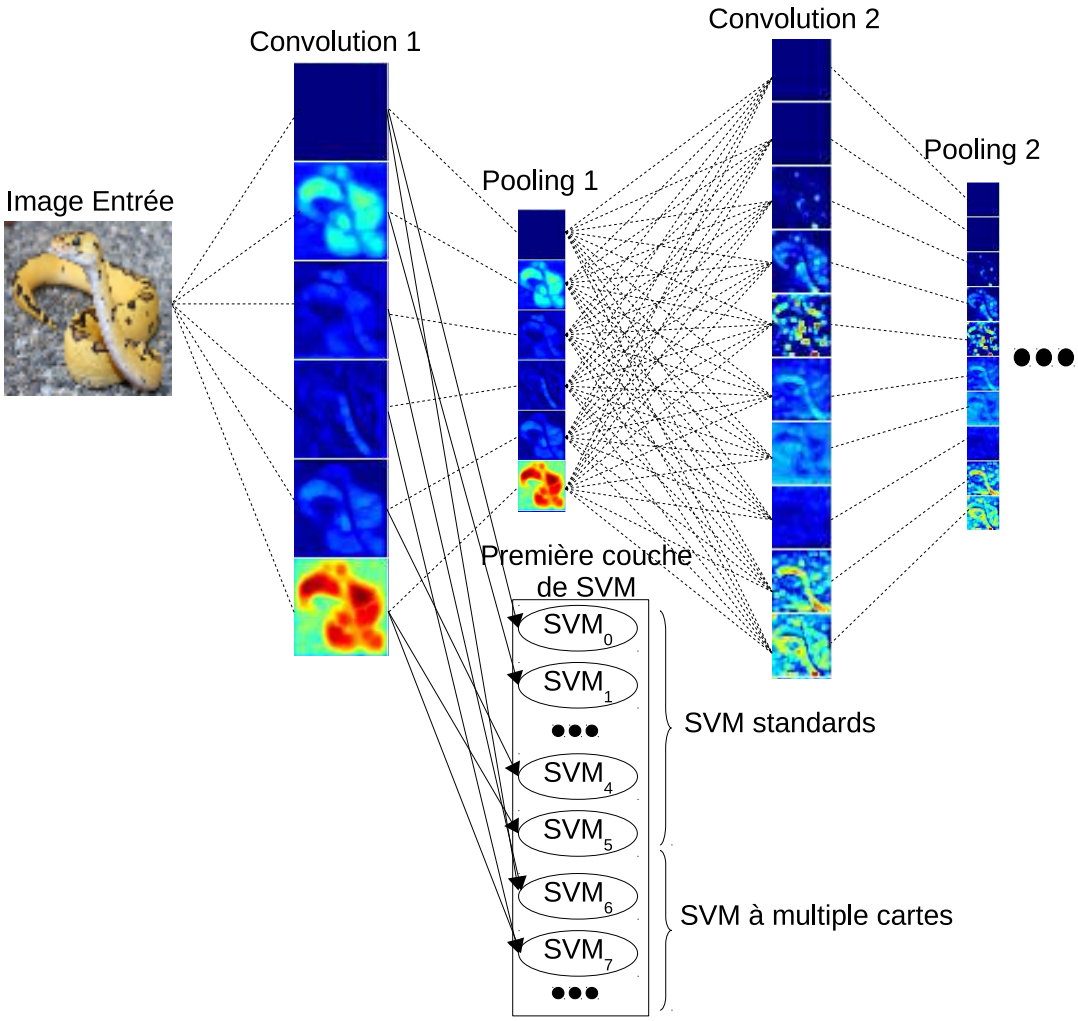


FIGURE 8.4 – Représentation d'une couche de SVM qui est connectée à la première couche d'un CNN. Au sein de la couche de SVM, deux SVM sont représentés : les SVM standards et les SVM multiples se connectant à deux cartes de caractéristiques.

L^{SVM6} prend en entrée la couche L^{SVM4} et L^{SVM5} et la couche L^{SVM9} prend en entrée la couche L^{SVM6}, L^{SVM7} et L^{SVM8}.

Afin d'augmenter la diversité dans le réseau, nous avons choisi de rajouter 20, 40 et 60 SVM à multiples cartes aux trois premières de couche de SVM. Chaque SVM à multiples cartes se connecte à 6 cartes de caractéristiques aléatoires dans la couche associée du CNN. Les couches L^{SVM1}, L^{SVM3} et L^{SVM5} contiennent donc 68, 232 et 348 SVM respectivement. Ce choix empirique peut être optimisé en étant déterminé par une base de validation, mais le coût d'apprentissage est alors très élevé. Les couches cachées permettent de réduire et de condenser l'information des couches précédentes. Nous notons $N^{(c)}$ le nombre de SVM sur la couche c et $M^{(c)}$ le nombre de connexions possédées par les SVM d'une couche c. Si la couche c est une couche d'entrée alors $M^{(c)}$ représente le nombre de cartes de caractéristiques prises en entrée par chaque SVM. Dans le cas où c est une couche cachée alors $M^{(c)}$ représente le nombre de scores prove-

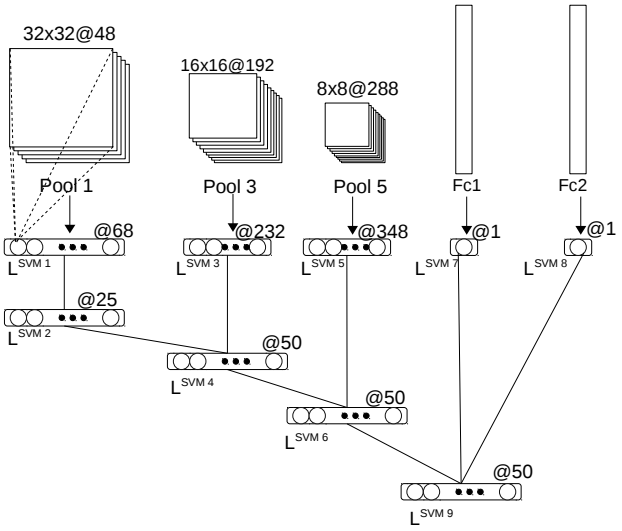


FIGURE 8.5 – Architecture du réseau de SVM avec les points d'ancrage au réseau CNN.

nant des SVM des couches prises en entrées. Le tableau 8.2 décrit en détail les paramètres des couches du réseau de SVM.

c	$N^{(c)}$	$M^{(c)}$	c	$N^{(c)}$	$M^{(c)}$
SVM1	68	1 et 6	SVM2	25	30
SVM3	232	1 et 6	SVM4	50	50
SVM5	348	1 et 6	SVM6	50	70
SVM7 et SVM8	1	4096	SVM9	50	70

TABLE 8.2 – Détails des paramètres des couche pour le réseau de SVM proposée.

8.3 Évaluation expérimentale

8.3.1 Évaluation des performances du CNN avec un unique SVM

La figure 8.6 montre la courbe ROC en bleu du réseau profond CNN ayant effectué un apprentissage multi-classes. Ce réseau a des performances très inférieures, de l'ordre de 50% de précision en moins, à celles d'un réseau de SVM utilisant des descripteurs HOG calculés sur des images en niveau de gris. La succession de convolutions et de sous-échantillonnages dans le CNN rendent le réseau invariant à de nombreuses transformations rigides, notamment aux translations. Autrement dit, il n'arrive pas à distinguer les parties de tombes aux tombes entières, ce qui provoque un mauvais placement de la fenêtre englobante sur la tombe et donc des performances bien inférieures.

La courbe rouge de la figure 8.6 représente les performances d'un SVM unique ayant effectué son apprentissage sur la dernière couche du CNN. Il s'agit du SVM présent sur la couche L^{SVM8} du réseau de SVM. Nous constatons alors un gain considérable de performance. Le SVM permet maintenant de correctement centrer les fenêtres englobantes et augmente la précision de 5% par rapport à un réseau de SVM+HOG pour un rappel de 75%.

En comparant le CNN+SVM unique à l'architecture d'un réseau de SVM empilée, présentée dans la section 6.3.2, on constate que pour un rappel inférieur à 73%, les performances sont similaires. Nous pouvons en déduire que l'apprentissage par rétro-propagation du CNN extrait correctement les motifs importants présents dans la couleur. Ceci dit, pour un rappel supérieur à 73%, les deux approches chutent rapidement. Le réseau de SVM empilé décroit plus rapidement, puisque pour un rappel de 80%, la précision est de 50% contre 68% pour le CNN+unique SVM. Finalement, les deux méthodes ont des résultats inexploitables pour un rappel supérieur à 85%.

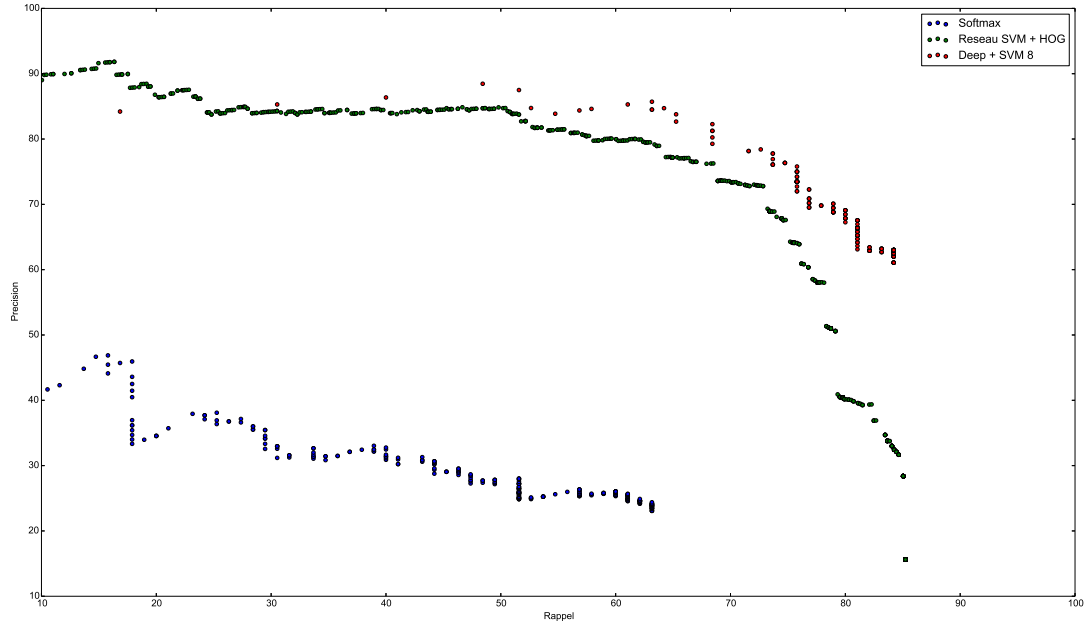


FIGURE 8.6 – Représentation de trois courbes ROC correspondantes aux performances d'un CNN seul (bleu), d'un CNN+SVM unique (rouge) et d'un réseau de SVM avec des HOG (vert).

Dans cette section nous avons montré que le couplage CNN+unique SVM est plus efficace que l'utilisation d'un seul CNN ou que l'utilisation d'un réseau de SVM avec des HOG. Le coût d'activation d'un réseau de neurones convolutifs est cependant beaucoup plus important que celui nécessaire pour extraire les caractéristiques HOG. Nous allons donc incorporer le chemin d'activation dans le CNN combiné avec un réseau de SVM pour réduire les temps de calculs.

8.3.2 Évaluation des performances du chemin d'activation

8.3.2.1 Implémentation du chemin d'activation dans les CNN

Afin d'effectuer les apprentissages et les évaluations des réseaux CNN nous utilisons le *framework CAFFE* [59]. Celui-ci propose une implémentation très optimisée des architectures CNN. Il utilise notamment la librairie CuDNN [1] qui est une implémentation Nvidia qui permet d'effectuer des calculs propres à l'apprentissage profond. Cette architecture utilise le concept de paquets (vu dans la section 4.2.3) d'images pour l'optimisation des calculs, ce qui provoque des problèmes d'utilisation avec le chemin d'activation. En effet, lors de la phase d'évaluation, le réseau de SVM doit également traiter des paquets d'images. Ainsi, lors de l'activation, certains SVM du chemin d'activation seront confiants avec certaines images du paquet et pas avec d'autres, ce qui nécessite de modifier dynamiquement la taille des paquets. Sur la figure 8.7 nous illustrons ce problème. On peut voir un paquet de 6 images transmises : 5 images de papillons et une image d'ours blanc. Le résultat de la première couche de convolutions est transmise à la couche d'entrée du réseau de SVM. Certains neurones SVM du réseau sont alors confiants avec l'image de l'ours blanc. Il est donc nécessaire de modifier la taille du paquet transmis à la couche de sous-échantillonnage et donc au reste du réseau. Il n'est pas possible de prévoir à l'avance combien d'images du paquet seront reconnues par des SVM de confiance et donc il est nécessaire d'adapter la taille du paquet dynamiquement après l'activation de chaque couche de neurones SVM.

Adapter la taille des paquet n'est pas trivial à implémenter et doit être effectué sans perte de temps. Pour résoudre ce problème, nous proposons deux architectures différentes :

— Une première solution, très simple, nécessite peu de modifications du code CAFFE. Elle consiste à utiliser des paquets d'une seule image. Les couches vont ainsi s'activer les unes après les autres dans l'ordre défini par le chemin d'activation. Si une condition de rejet ou d'acceptation de l'image a lieu durant l'activation, le réseau peut stopper l'activation. Cependant, le gain éventuel n'est pas représentatif puisque nous avons activé le réseau pour une seule image alors que l'activation pour plusieurs images aurait été plus optimisée. Aucune véritable comparaison ne peut donc être faite.

— Une seconde solution considère une phase d'évaluation classique où des paquets de plusieurs images sont transmis au réseau. Durant l'activation, si une condition de rejet ou d'acceptation a lieu, il est nécessaire d'ajuster la taille des paquets pour éliminer les images dont on connaît la classe. Effectuer ce changement de taille, sans perte d'optimisation et

1. https ://developer.nvidia.com/cudnn

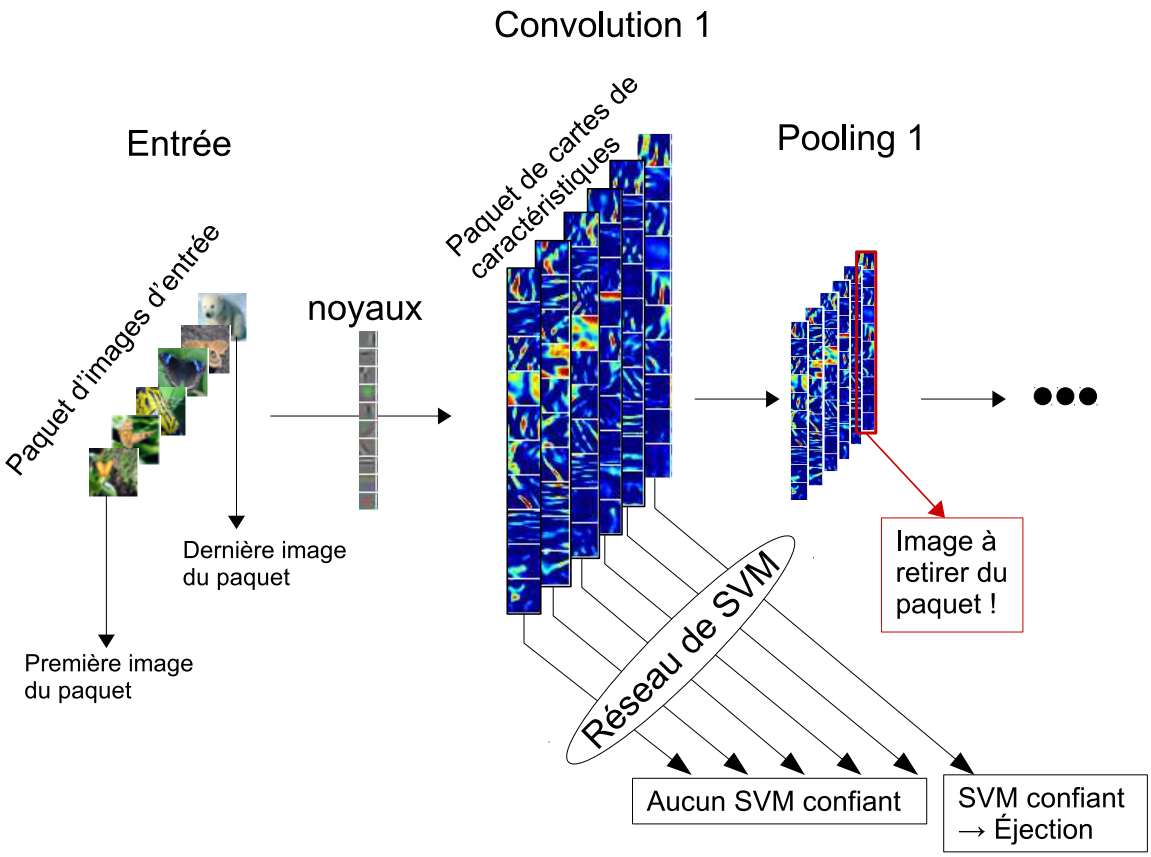


FIGURE 8.7 – Illustration de l'activation du CNN combiné avec un réseau de SVM. Un paquet de 6 images est transmis au réseau. Dans ce paquet, l'image représentant un ours blanc est reconnue par le réseau de SVM avec un haut niveau de confiance. Elle doit donc être retirée du paquet, ce qui modifie dynamiquement la taille du paquet pour la couche *Pooling1* dans le CNN.

de manière dynamique au sein du réseau, est complexe. Aussi, une solution consiste à effectuer l'activation avec des paquets dont la taille est toujours la même. Pour cela, lorsqu'un paquet est complet il va activer sa couche. Si jamais des éjections d'images se sont produites, alors nous stoppons l'activation de la couche et passons en entrée un nouveau paquet d'images venant compléter le premier. Lorsque celui-ci est complété l'activation du réseau reprend. Avec cette solution les couches s'activent toujours avec un paquet complet et donc aucune perte de temps de calculs liée aux paquets n'a lieu. Cependant mettre en place cette architecture est complexe et demande de grandes connaissances en programmation CUDA ainsi que sur le framework CAFFE.

Nous avons uniquement implémenté la première solution, ce qui nous empêche de comparer les vrais gain en temps de calculs. Nous allons donc tenter d'évaluer théoriquement ces gains calculatoires.

8.3.2.2 Évaluation du temps de calculs du CNN

Pour évaluer notre chemin d'activation, nous définissons une fonction de coût, notée $\Omega^{(c)}$, tel que $\Omega^{(c)}$ est le coût d'activation de la couche c. Pour construire cette fonction, nous négligeons le coût de la fonction d'activation, le coût des couches de sous-échantillonnage et des couches de normalisation.

Ainsi cette fonction évalue uniquement le nombre de calculs effectués par les couches *intelligentes* de la façon suivante :

$$\Omega^{(c)} = \Omega^{(c-1)} + \begin{cases} |\mathbf{w}^{(c)}| \times out^{(c-1)} \times N^{(c)} & \text{Si convolution} \\ out^{(c-1)} \times N^{(c)} & \text{Si entièrement connecté} \\ 0 & \text{Sinon} \end{cases} \quad (8.1)$$

avec c une couche prenant en entrée la couche $c-1$ et $out^{(c)}$ une fonction retournant la taille du vecteur de sortie de la couche définie par :

$$out^{(c)} = \begin{cases} width^{(c-1)} \times height^{(c-1)} \times N^{(c)} & \text{Si convolution} \\ N^{(c)} & \text{sinon} \end{cases} \quad (8.2)$$

avec $width^{(c-1)}$ et $height^{(c-1)}$ la largeur et hauteur des cartes de caractéristiques de la couche $c-1$.

A partir de la fonction coût Ω, nous calculons le coût relatif à chaque couche du réseau. Nous notons $RC^{(c)}$ le coût relatif de la couche c défini par l'équation 8.3.

$$RC^{(c)} = \frac{\Omega^{(c)}}{\Omega^{(FC2)}} \quad (8.3)$$

A l'aide du calcul 8.4 nous mettons en évidence le fait que le coût d'activation des couches de neurones de type SVM est négligeable devant celui des neurones convolutifs. En effet, si nous prenons l'exemple de la couche L^{SVM1} il est 1000 fois moins important que celui de la couche L^{CONV2}.

$$\begin{aligned} \Omega^{(SVM1)} &= 32 * 32 * (48 + 6 * 20) + \Omega^{(CONV1)} \\ \Omega^{(CONV2)} &= 32 * 32 * 48 * (5 * 5 * 96) + \Omega^{(CONV1)} \\ \Rightarrow \frac{\Omega^{(SVM1)} - \Omega^{(CONV1)}}{\Omega^{(CONV2)} - \Omega^{(CONV1)}} &= 0.001458 \end{aligned} \quad (8.4)$$

Le tableau 8.3 montre le coût relatif des différentes parties du réseau. Nous constatons que le coût calculatoire est important à la couche $Conv3$ où le coût relatif augmente de 0.5. Aussi, il serait donc intéressant d'effectuer des rejets avant cette couche, ce que les couches $SVM1$ et $SVM2$ peuvent effectuer. Cependant, les SVM de la couche $SVM1$ ne prennent en entrée que des images une seule fois convoluées. Le niveau d'abstraction est alors très faible et seuls des objets relativement simples pourront être reconnus.

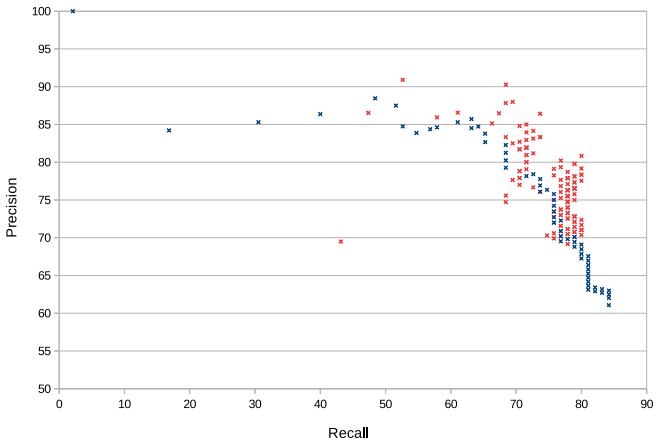
Dans la sous-section suivante, nous allons évaluer les performances et le gain calculatoire théorique induit par le chemin d'activation sur un réseau combinant des neurones convolutifs et des SVM.

c	Conv1	Conv2	Conv3	Conv4	Conv5	FC2
$RC^{(c)}$	0.031	0.159	0.670	0.762	0.90	≅1

TABLE 8.3 – Coût relatif des différentes couches du réseau CNN utilisé.

8.3.2.3 Évaluation du couplage CNN et réseau de SVM

Le réseau de SVM comporte le nombre important de 825 SVM (cf tableau 8.2) apprenant sur des concepts plus ou moins avancés. Les premiers SVM utilisent uniquement des cartes de caractéristiques filtrées qui sont des caractéristiques très peu robustes. Nous fixons la valeur de p_{min} (cf chapitre 7) à 0.995, afin de fixer les seuils de confiance pour la détection des objets de classes : *tombes* et *non tombes*, pour tous les SVM. De plus, afin de gagner en robustesse nous faisons varier la valeur de θ_{lim}. Sur la figure 8.8.a nous



(a)

θ_{lim}	Rappel	Précision	RC (%)
0	43.15	69.4915	1.38
1	47.36	86.5385	1.59
2	52.63	90.90	1.75
5	68.42	90.27	2.17
10	70.52	84.81	2.72
11	70.52	82.7	2.83
26	71.57	80.95	4.74
27	70.52	77.90	4.97

(b)

FIGURE 8.8 – Le graphique de gauche montre les courbes ROC des approches : CNN+unique SVM en bleu et CNN+réseau de SVM en rouge. Les performances du CNN+réseau de SVM sont données pour des valeurs de θ_{lim} variable. Le tableau reprend les performances du réseau de SVM avec différents θ_{lim} et met en évidence les coûts relatifs d'activation nécessaires.

comparons les performances d'un CNN+unique SVM et d'un CNN+réseau de SVM. Nous constatons que le réseau de SVM a des performances très variables en fonction de son paramètre θ_{lim}. Comme annoncé dans la section précédente, le tableau 8.8.b montre que plus θ_{lim} augmente, plus les performances s'améliorent. Par exemple pour $\theta_{lim} = 0$ la F_{mesure} est de 53.24% contre 77% avec un $\theta_{lim} = 10$. Afin de déterminer au mieux ce paramètre de seuil, nous effectuons une évaluation sur une base validation.

En utilisant les résultats présentés sur la figure 8.9 nous constatons que plus θ_{lim} est grand plus la précision augmente jusqu'à converger vers une valeur maximale de 94% pour θ_{lim} supérieur à 15. De plus sur la base de validation, la figure 8.9 montre que le rappel évolue en deux temps. Dans un premier temps

il augmente jusqu'à $\theta_{lim} = 5$, puis, il décroît progressivement. Cette évolution est normale et s'explique par le fait qu'il est de plus en plus difficile de confirmer la présence ou l'absence d'un objet lorsque θ_{lim} est trop grand. En effet, lorsque θ_{lim} devient trop important, il n'y a plus assez de SVM pour confirmer la présence d'un objet, donc le nombre de preuves accumulées P n'atteint jamais θ_{lim}, provoquant une baisse de rappel. Sur cette base de validation, les meilleures performances sont atteintes pour $\theta_{lim} = 5$ avec une *Fmesure* d'environ 60%. Nous utilisons donc cette valeur pour la phase d'évaluation finale.

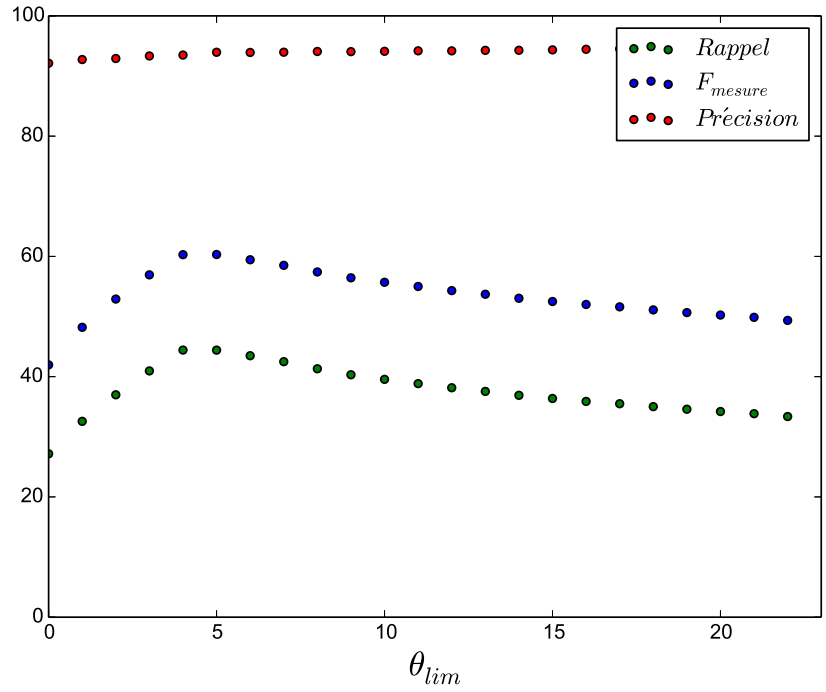


FIGURE 8.9 – Performances du CNN+réseau de SVM sur base de validation en fonction du paramètre θ_{lim}.

Lors de la phase d'évaluation, en prenant $\theta_{lim} = 5$ nous obtenons une précision de 90.2% pour un rappel de 68.4%. Pour ce même rappel, le CNN+unique SVM a une précision de 82%, soit un gain de 8% en précision pour le CNN+réseau de SVM. Ce gain nous permet d'affirmer que les réseaux de SVM combinés aux CNN permettent une augmentation significative des performances.

Nous nous sommes intéressés à l'évolution du seuil θ_{lim} en fonction de la précision. Nous allons analyser le comportement de la précision en fonction du seuil θ_{lim} , puis nous testerons si l'allure des résultats suit la loi prédite dans l'équation 7.4. Sur la figure 8.10, nous constatons que la précision augmente rapidement avec le nombre de SVM confiants requis θ_{lim}. Ce résultat est en accord avec la prédiction de l'équation 7.4 et peut s'ajuster par une équation de la forme $ax^b + c$. Cependant, après une valeur pallier la précision chute. Nous modélisons ce phénomène par une équation de la forme : $a(b + x)^c$. Nous

cherchons alors par une méthode des moindres carrés, à modéliser l'évolution de la précision par l'équation suivante :

$$a_1 x^{a_2} + a_3(a_4 + x)^{a_5} + a_6 \tag{8.5}$$

avec $a_1, a_2, a_3, a_4, a_5, a_6$ des coefficients déterminés pour minimiser l'erreur quadratique. A l'aide de la dérivée première on trouve que le pic de performances est en $\overline{\theta}_{lim} = 2.19$. On constate que tant que $\theta_{lim} < \overline{\theta}_{lim}$, la précision augmente très rapidement et la prédiction théorique est respectée. Pour $\theta_{lim} > \overline{\theta}_{lim}$, la tendance s'inverse et la précision chute de manière progressive. Il faut noter que la valeur $\theta_{lim} = 2$ permet d'obtenir les meilleures performances, contrairement à la base de validation avec laquelle nous trouvons $\theta_{lim} = 5$. La recherche du meilleur paramètre, en utilisant la base de validation, n'est donc pas optimale. Ce problème peut être causé par la taille de la base de validation ne reflétant pas toute la diversité de l'objet recherché. Il serait alors intéressant d'augmenter la taille de cette base pour résoudre ce problème.

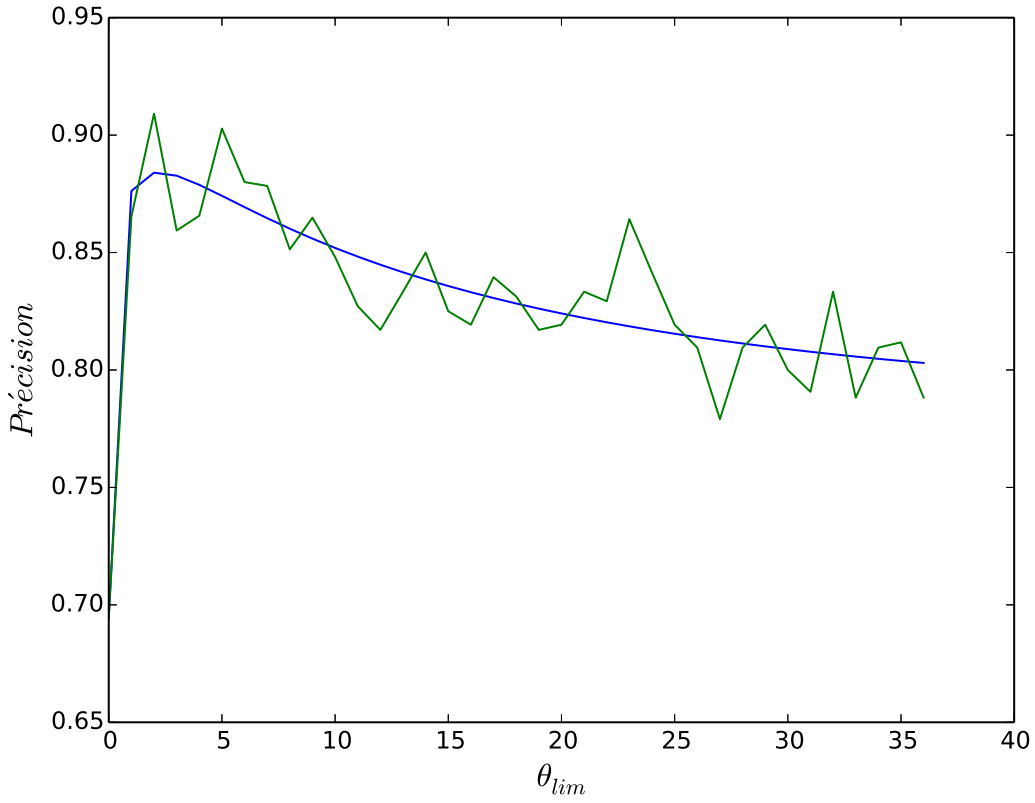


FIGURE 8.10 – La courbe verte représente la précision en fonction du seuil θ_{lim} . La courbe bleue est une courbe estimée par régression quadratique de la précision et est définie par : $0.23x^{0.26} - 611(3.93 + x)^{3.4\ 10^{-4}} + 612$.

Dans un second temps nous nous focalisons sur la réduction potentielle des coûts de calculs. Le tableau 8.8.b exprime le coût calculatoire relatif en fonction de θ_{lim}. Le coût calculatoire est inversement proportionnel aux nombres

de couches utilisées. En effet, pour $\theta_{lim} = 1$ seulement 1.38% des couches du réseau sont utilisées alors que 5% le sont pour $\theta_{lim} = 27$. Pour la valeur de θ_{lim} déterminée par la base de validation, on constate que seulement 2.17% du réseau s'active. En moyenne plus de 97.8% des calculs effectués, sur les couches profondes du réseau, n'ont pas d'utilité. Le gain potentiel est donc très significatif.

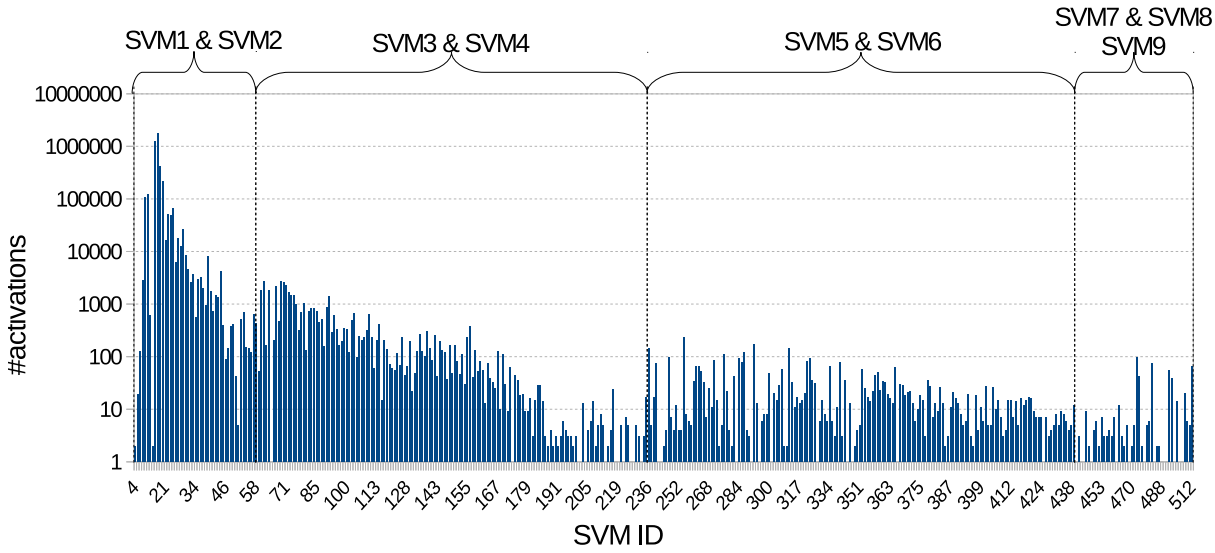


FIGURE 8.11 – Représentation du nombre d'activations de chaque SVM à l'intérieur du réseau. Une activation a lieu lorsqu'un SVM accepte ou rejette la fenêtre courante.

Sur l'histogramme de la figure 8.11 nous représentons la fréquence d'activation des différents SVM du réseau. Comme attendu, plus un SVM est connecté à une couche profonde, moins il s'active. En effet, pour atteindre une couche profonde l'objet doit être "complexe", puisqu'il n'a pas été reconnu par les couches précédentes. Par exemple les SVM présents sur les couches $SVM1$ et $SVM2$ s'activent en moyenne 100 fois plus que ceux présents sur les couches $SVM3$ et $SVM4$. Nous notons que peu de SVM restent inactifs, ce qui montre que le nombre total de SVM n'est pas trop élevé et que les SVM présentent peu de redondances.

Afin de mieux évaluer le réseau de SVM le tableau 8.4 permet de représenter le taux d'erreurs lorsque la condition d'arrêt est atteinte. On constate que les premières couches du SVM ont une probabilité d'erreurs très faible. Cependant, elles servent surtout à filtrer les fenêtres candidats ne possédant pas de tombes, car seul 0.02% des tombes y sont détectées. Plus la couche analysée est profonde, plus le nombre d'objets correctement classés augmente car les SVM traitent l'information avec de plus en plus de descripteurs. On remarque que les couches $SVM5$ et $SVM6$ ont un taux d'erreur important. La couche $SVM5$ contient 348 SVM dont la plupart apprennent uniquement à partir d'une carte de caractéristiques de dimensions 8×8. L'information transmise à ces SVM est

c	#Activations	Erreur (%)	FP (%)	TP (%)
L^{SVM1} et L^{SVM2}	3 946 414	0.38	0.38	0.02
L^{SVM3} et L^{SVM4}	31 433	18.76	18.61	0.71
L^{SVM5} et L^{SVM6}	3 724	50.70	49.03	5.89
L^{SVM1}, L^{SVM1} et L^{SVM2}	450	42.85	30.84	35.61

TABLE 8.4 – Le tableau suivant représente les performances des couches lors de leur activation, c'est-à-dire lorsqu'une condition d'arrêt d'exploration est atteinte. Les valeurs de FP et TP représentent le pourcentage d'activation provoquant des faux positifs et vrais positifs.

donc minime ce qui rend les SVM moins robustes. De plus, la couche $SVM5$ possédant de nombreux SVM, la condition d'arrêt est très vite atteinte et peut provoquer une erreur. Dans cette couche, il serait intéressant de remplacer les SVM d'entrées de type standard par des SVM à multiples cartes prenant donc plus d'informations en considération.
Pour lutter contre ce phénomène, il peut être intéressant de pondérer le vote des SVM de confiance. La pondération peut être proportionnelle à un score calculé sur une base de validation ou au niveau de confiance du SVM lors du vote. Pour cela, nous pouvons utiliser une fonction $f_i^{(c)} \in [0..1]$ qui représente le niveau de confiance du vote du SVM numéro i sur la couche c, ce qui permet de modifier l'équation 7.2 en l'équation 8.6. Par manque de temps, nous n'avons pas évalué expérimentalement cette métrique.

$$\begin{cases} p_i^{(c)} > \theta^{(a)}{}_i^{(c)} & \text{alors } P = P - f_i^{(c)} \\ p_i^{(c)} < 1 - \theta^{(b)}{}_i^{(c)} & \text{alors } P = P + f_i^{(c)} \\ \text{sinon} & SVM_i^{(c)} \text{ pas confident} \end{cases} \tag{8.6}$$

8.4 Fonction d'activation adaptative

8.4.1 Proposition d'une fonction adaptative

Dans cette section nous décrivons un travail en marge des réseaux de SVM et des chemins d'activation. Le réseau de neurones convolutifs est constitué de milliers de neurones, chacun possédant une fonction d'activation. Nous nous posons la question : comment choisir la meilleure fonction d'activation ? En effet, ce choix est arbitrairement fait par l'utilisateur lors de la construction du réseau. Dans la section 4.2.2, nous avons vu que beaucoup d'études mettent en évidence que la fonction d'activation influence grandement l'efficacité d'un réseau.

Nous proposons donc une fonction permettant au réseau de choisir automatiquement la meilleure fonction d'activation possible. Pour cela, nous considérons un ensemble de fonctions considérées comme étant performantes,

à savoir les fonctions : ELU, PReLU, Sigmoïde (Sigmoid) et tangente hyperbolique (tanH). Le tableau 8.5 résume les formes de ces fonctions.

Nom	Définition	Intervalle
ELU	$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$	$[-\alpha, +\infty[$
PReLU	$\begin{cases} x & x \geq 0 \\ \beta x & x < 0 \end{cases}$	$]-\infty, +\infty[$
Sigmoid	$\frac{1}{1+e^{-x\lambda}}$	$[0, 1]$
tanH	$tanh(x)$	$[-1, 1]$

TABLE 8.5 – Définitions des fonctions d'activation utilisées ainsi que leurs intervalles de valeurs.

Ces fonctions ont des avantages bien connus comme par exemple la convergence rapide des fonctions ELU et PReLU car leurs fonctions de transfert ne sont pas atténuées. Les fonctions Sigmoid et tanH permettent de lisser et de borner les signaux. Nous définissons alors la fonction, notée σ, qui combine ces fonctions avec des poids d'importance :

$$\sigma(x) = (\epsilon_1 ELU(x) + \epsilon_2 PReLU(x) + \epsilon_3 Sigmoid(x) + \epsilon_4 tanH(x)) \frac{1}{\sum_i \epsilon_i} \quad (8.7)$$

avec ϵ_1, ϵ_2, ϵ_3 et ϵ_4 les poids de pondération respectifs des fonctions : ELU, PReLU, Sigmoid et tanH.

Une des difficulté réside dans le fait que les fonctions utilisées retournent des valeurs qui ne sont pas directement comparables. Par exemple, lorsque $x = 5$, $tanH(5) = 0.999$ alors que les fonctions ELU et $PReLU$ retournent 5. Nous modifions alors la vitesse de convergence des fonctions ELU et $PReLU$ afin d'obtenir des valeurs moins importantes quand x est grand. Nous avons donc modifié les fonctions ELU et PReLU pour qu'elles convergent moins rapidement vers l'infinie quand x est grand. Pour cela nous remplaçons la fonction : $x \to x \; \forall x > 0$ par $x \to \begin{cases} \sqrt{x} & \forall x > 1 \\ x & \forall x \in [0..1] \end{cases}$. La fonction $\sigma(x)$ peut donc s'exprimer suivant l'équation 8.8 :

$$\sigma(x) = \left(\epsilon_1 \begin{cases} \sqrt{x} & x \geq 1 \\ x & 0 \leq x < 1 \\ \alpha(e^x - 1) & x < 0 \end{cases} + \epsilon_2 \begin{cases} \sqrt{x} & x \geq 1 \\ x & 0 < x < 1 \\ \beta x & 0 > x > -1 \\ -\beta\sqrt{-x} & x \leqslant -1 \end{cases} + \epsilon_3 \frac{1}{1+e^{-x\lambda}} + \epsilon_4 tanh(x) \right) \cdot \frac{1}{\sum_i \epsilon_i} \quad (8.8)$$

L'objectif de notre fonction est d'automatiser la détermination des meilleurs poids de pondération par le réseau. Pour cela, les poids se mettent à jour lors

de l'apprentissage du réseau par un processus de rétro-propagation. Pour effectuer cela, comme indiqué dans la section 4.2.1, nous avons besoin des dérivées données par :

$$\frac{\partial \sigma}{\partial \epsilon_i} = \left((\sum_j \epsilon_j - \epsilon_i).f_i(x) - \sum_{j=\{1,2,3,4\}\setminus i} (\epsilon_j.f_j(x)) \right) . \frac{1}{\left(\sum_j \epsilon_j\right)^2}$$

avec f_i la fonction associée au poids ϵ_i.

Pour intégrer cette fonction au réseau et au mécanisme de rétro-propagation il est nécessaire de calculer la fonction de transfert associée. Cette fonction de transfert permet la régularisation de l'erreur transmise aux couches parents et correspond à la dérivée de la fonction d'activation, c'est à dire :

$$\frac{\partial \sigma}{\partial x} = \left(\epsilon_1 \begin{cases} \frac{1}{2\sqrt{x}} & x \geq 1 \\ 1 & 0 \leq x < 1 \\ \alpha e^x & x < 0 \end{cases} + \epsilon_2 \begin{cases} \frac{1}{2\sqrt{x}} & x \geq 1 \\ 1 & 0 < x < 1 \\ \beta & 0 > x > -1 \\ \beta \frac{1}{2\sqrt{-x}} & x \leqslant -1 \end{cases} + \epsilon_4(1 + tanh(x)^2) + \epsilon_3 \gamma \frac{e^{-x\gamma}}{(1+e^{-x\gamma})^2} \right) . \frac{1}{\sum_i \epsilon_i} \tag{8.9}$$

À partir de la fonction de transfert donnée dans l'équation 8.9, celle-ci devient facilement implémentable dans le framework CAFFE. Nous donnons les résultats de l'évaluation dans la prochaine sous-section.

8.4.2 Évaluation de la fonction adaptative

Pour évaluer l'efficacité de notre fonction adaptative nous avons utilisé deux bases de référence à savoir :

— MNIST, une base de reconnaissance de lettres,
— CIFAR100 [64], une base de classification d'images contenant des animaux.

Le réseau utilisé se compose de 3 couches de convolutions connectées à 2 couches de sous-échantillonnage. Les deux premières couches de convolutions sont précédées de deux couches de normalisation LRN. La dernière convolution est connectée à deux couches entièrement connectées qui se composent de 4096 neurones chacune. L'objectif de ce réseau de petite taille n'est pas d'obtenir les meilleurs performances mais de comparer les fonctions d'activation entre-elles. Pour cela, les résultats présentés dans le tableau 7.1 sont issus d'une validation croisée répétée trois fois.

Nous constatons que le PReLU surpasse systématiquement le ReLU d'environ 1% à 2% pour les bases MNIST et CIFAR. Notre fonction adaptative a de meilleurs résultats avec un gain de 2% par rapport au PReLU sur la base CIFAR et un gain de 0.76% pour la base MNIST.

	ReLU	PReLU	Notre fonction
MNIST	97.34%	98.41%	99.17%
CIFAR 100	58.32%	60.25%	62.98%

TABLE 8.6 – Comparaisons des taux de classification corrects des différentes fonctions d'activation sur une base de référence avec des architectures de réseaux identiques.

Ce gain de performances est important et il serait intéressant d'étudier la répartition des poids affectés à chaque fonction dans le réseau. En plus d'avoir des fonctions d'activation, se déterminant de manière automatique, nous aimerions élargir ce fonctionnement aux couches de sous-échantillonnages. Nous travaillions actuellement sur des couches dont le résultat est une pondération entre un sous-échantillonnage moyen et un sous-échantillonnage par le max. Ces réflexions font l'objet de certains de nos travaux actuels.

8.5 Bilan des contributions sur les CNN

Notre proposition majeure avait un double objectif et consistait à combiner un réseau de neurones convolutifs avec un réseau de SVM. Le premier fut d'améliorer les performances à l'aide d'un traitement plus fin de la résolution par le réseau de SVM. Le second fut de permettre l'utilisation du chemin d'activation afin de réduire le coût d'activation d'un réseau et de permettre une application plus rapide. L'évaluation de notre approche CNN+réseau de SVM a montré un gain de 8% en précision pour un rappel d'environ 70% [86]. De plus, le chemin d'activation permettrait d'éviter potentiellement 97.7% des calculs en stoppant l'activation du réseau de SVM et donc l'activation du CNN. Pour obtenir ce gain calculatoire, il serait pertinent de nous intéresser à l'implémentation de CAFFE afin de proposer une méthode optimisée permettant la gestion de paquets à tailles dynamiques.

Dans un second temps nous avons proposé une fonction d'activation adaptative permettant d'accroître les performances. En effet, l'utilisateur choisit les fonctions d'activation utilisées dans le réseau mais sans garantie qu'elles seront les plus performantes. Notre fonction adaptative renvoie la valeur pondérée de plusieurs fonctions connues avec des poids déterminés par rétro-propagation. Cette fonction permet une réduction de l'erreur de 1 à 3% sur des bases de référence tels que CIFAR10 et CIFAR100. Elle peut tout à fait, être intégrée dans le réseau de convolutions utilisé par le réseau de SVM afin d'obtenir de meilleures performances.

Chapitre 9

Conclusions et perspectives

Nous allons dans une première section reprendre l'ensemble de nos contributions et faire un bilan du travail effectué. Puis, dans une seconde section, nous donnerons quelques pistes à améliorer et des idées pour y parvenir.

9.1 Bilan des recherches

Dans cette thèse nous avons proposé une nouvelle architecture basée sur des réseaux de SVM afin de mieux prendre en compte la résolution des descripteurs. Dans un premier temps, nous avons combiné notre réseau de SVM avec des descripteurs multi-résolutions HOG extraits sur des images en niveaux de gris. Dans ce contexte, nous avons montré que l'utilisation d'un réseau de SVM permettait une augmentation de la précision de plus de 10% par rapport à un SVM unique. Par la suite, nous avons mis en évidence que le réseau de SVM peut fusionner différents types de données en entrée. Nous nous sommes alors intéressés à l'information couleur et à la façon de mieux la considérer à l'aide de HOG extraits sur les trois composantes couleurs. En outre, nous avons constaté que le réseau permet un gain assez faible de 3% comparé à une approche n'utilisant qu'un seul SVM. Nous avons proposé et comparé différentes architectures de réseaux de SVM et avons mis en évidence l'architecture *empilée*. Pour chaque résolution, les différentes caractéristiques sont normalisées, puis l'ensemble des résolutions est fusionné. Elle permet un gain considérable de l'ordre de 10% en précision par rapport à un SVM simple.

Le réseau de SVM, bien que performant, est cependant très coûteux en calculs. Pour une utilisation courante, il est nécessaire de réduire le temps de traitement lors de la phase d'évaluation. Pour cela, nous avons introduit le chemin d'activation qui définit un ordre d'activation des SVM dans le réseau et qui stoppe l'activation lorsqu'un nombre suffisant de SVM est confiant avec une décision. Afin de gagner en performances, nous avons rajouté en plus un critère de cumuls de preuves. En effet, lorsqu'un SVM est confiant, l'exploration ne va pas se stopper instantanément mais une preuve va être accumulée en faveur

de la classe prédite. Lorsque le nombre de preuves est suffisamment important, nous stoppons l'activation du réseau. Le chemin d'activation, appliqué à des réseaux de SVM combiné avec des descripteurs HOG, permet de réduire les temps de calculs d'un facteur cinq. De plus, contrairement à une cascade de classifieurs, la précision augmente jusqu'à 1% pour un large intervalle de valeurs de rappels. Le chemin d'activation pourrait permettre une utilisation quasiment en temps réel du système de détection.

L'ensemble des travaux présenté ci-dessus considère deux phases distinctes : une première phase d'extraction de caractéristiques et une seconde phase de classification. Grâce au gain calculatoire procuré par les cartes graphiques ces dernières années, les méthodes d'apprentissages profonds font l'objet de nombreuses recherches. Nous avons donc voulu comparer nos résultats à ces approches. Pour cela, nous avons montré dans un premier temps l'importance d'avoir une base d'apprentissage variée avec de nombreuses représentations des objets afin que le réseau crée un modèle générique. Nous avons proposé une fonction d'activation intelligente permettant de déterminer automatiquement par rétro-propagation la fonction la plus adaptée. Ainsi une couche peut être influencée par une pondération de plusieurs fonctions connues comme la tangente hyperbolique et le ReLU. Puis, nous avons utilisé le réseau profond, à savoir un CNN, comme un extracteur de caractéristiques optimisées. A partir de ces caractéristiques un SVM est entraîné. Ce SVM obtient alors d'excellentes performantes qui rivalisent avec celles de l'architecture empilée du réseau de SVM.

Le SVM entraîné n'utilise finalement que la sortie d'une seule couche du CNN et donne de très bons résultats. Nous avons donc adapté le réseau de SVM pour qu'il se connecte au réseau de neurones convolutifs à différents niveaux de profondeur. L'objectif est d'une part d'obtenir un meilleur traitement de l'information extraite par le CNN et d'autre part de pouvoir mettre en place un système de rejet anticipé basé sur le chemin d'activation. Pour cela, à chaque niveau de profondeur, les cartes de caractéristiques sont utilisées et peuvent être vues comme des descripteurs d'une résolution. Une fois le réseau de SVM construit, afin de réduire le coût calculatoire, nous y avons intégré le chemin d'activation. A l'aide de la base de validation, nous avons trouvé le nombre optimal de SVM de confiance pour stopper l'exploration du chemin d'activation. Pour cette valeur, le réseau de SVM prenant en entrée un CNN a une précision accrue de 8% pour un rappel d'environ 70%. De plus, le chemin d'activation permet d'éviter l'activation de nombreuses couches du CNN. Au final, seulement 2.17% des calculs effectués par le CNN sont utilisés, et donc un gain de rapidité calculatoire allant jusqu'à 97.83% pourrait être obtenu avec une bonne implémentation. Économiser les calculs est très intéressant sur les cartes em-

barquées, telle que la *Jetson TX1* utilisée pour les systèmes embarqués et autonomes.

En conclusion, dans ces travaux de recherches nous avons apporté plusieurs solutions à notre problème de détection et de localisation. Bien que plus performantes, nos méthodes ne donnent pas pour le moment des performances suffisantes pour une automatisation de la tâche de localisation des tombes. En effet, nous obtenons pour le réseau de SVM utilisant la couleur avec une architecture empilée, une précision de 82% pour un rappel de 68.4%. Par ailleurs, pour le réseau de SVM utilisant des CNN nous obtenons une meilleure précision de 90.2% pour le même rappel.

9.2 Perspectives

Dans cette section nous listons plusieurs pistes d'améliorations des méthodes présentées dans la thèse.

Sur le court terme, il serait intéressant d'utiliser le réseau de SVM pour fusionner des descripteurs de différentes natures. Grâce au chemin d'activation, il est notamment possible d'utiliser un grand nombre de caractéristiques et de les extraire seulement si le neurone SVM les utilisant s'active. Ainsi, nous pouvons créer une architecture de type CNN combiné à un réseau de SVM où le réseau de SVM prend aussi en entrée des descripteurs photométriques, de textures et de contours.

Une seconde amélioration est liée à un problème dans l'apprentissage du réseau de SVM. Durant l'apprentissage, chaque SVM utilise un ensemble de caractéristiques unique qui est extrait sur la totalité de la base d'apprentissage. Or, si une représentation spécifique d'un objet est plus représentée qu'une autre dans la base, tous les SVM chercheront à la détecter. Par exemple, les concessions possédant une pierre tombale grise sur un fond de terre couleur orangée, seront parfaitement détectées par de nombreux SVM. Cependant, il est inutile que tous les SVM reconnaissent cette représentation. Pour éviter ce problème, plusieurs solutions sont envisageables.
La première est simple et consiste à effectuer l'apprentissage de chaque SVM sur une partie aléatoire de la base d'entraînement.
La seconde consiste à effectuer l'apprentissage des SVM dans le réseau de manière jointe avec le chemin d'activation. Après l'apprentissage de chaque couche du réseau de SVM, nous définissons alors le chemin d'activation. Puis, nous évaluons chaque image de la base d'apprentissage. Lorsqu'un nombre suffisant de SVM est confiant avec la reconnaissance d'une imagette celle-ci est supprimée de la base. Une fois ce processus effectué, nous continuons l'apprentissage de la couche de SVM suivante avec la base réduite. Cette seconde solution a comme majeur défaut le fait que le chemin se construise couche par

couche et ne possède donc pas de profondeur.
Pour éviter ce problème, une troisième solution très coûteuse peut-être envisagée lors de la phase d'apprentissage. Celle-ci consiste à effectuer l'apprentissage complet classique puis à construire un chemin d'activation temporaire. Ensuite, les N meilleurs SVM du chemin temporaire sont ajoutés à un chemin d'activation définitif. Chaque image de la base d'apprentissage est alors testée par les SVM contenus dans le chemin d'activation définitif. Si suffisamment de SVM sont confiants avec l'image, celle-ci est retirée de la base d'apprentissage. Puis nous re-effectuons l'apprentissage de l'ensemble des SVM qui ne sont pas dans le chemin d'activation définitif avec la base réduite. Ensuite nous répétons le processus, en construisant des chemins d'activation temporaires puis en ajoutant les N meilleurs SVM au chemin d'activation définitif. Une fois tous les SVM placés dans le chemin d'activation définitif l'apprentissage est fini.

Un troisième problème est le seuil de confiance. En effet, celui-ci est actuellement quantifié de façon binaire : soit le SVM dans le réseau de SVM est confiant, soit il ne l'est pas. Il peut être intéressant de nuancer la confiance du SVM comme présenté à la fin de la section 8.3.2.3. Pour cela, durant l'activation chaque SVM est considéré comme confiant et participe au cumule de preuves. Par exemple, le poids de la preuve pourrait être proportionnel à la probabilité de sortie du classifieur. Ainsi, pour les objets peu représentés dans la base de données (car très spécifiques), une probabilité finale serait donnée par l'ensemble des SVM du réseau de SVM. Cela peut permettre de détecter les tombes se confondant avec le sol et ne possédant pas de pierres tombales.

Enfin, pour améliorer les performances, de nombreuses règles peuvent aider à la classification. Dans cette thèse, nous n'avons fait aucun a priori afin de considérer le cas le plus général possible. Cependant les objets urbains sont souvent soumis à de nombreuses lois qui peuvent aider à la localisation et à la reconnaissance. Par exemple, pour la détection de plaques d'égouts, la loi fixe la distance maximale entre deux plaques à 80 mètres [1]. Cette contrainte peut permettre au modèle de détection de confirmer la présence d'un objet. En plus de l'information par contrainte, l'intégration de multiples informations fusionnées peut-être un avantage conséquent. Par exemple, en plus de l'information optique, l'information LiDAR permettrait d'obtenir la forme et la hauteur des objets observés. En 2015, Berger-Levrault a lancé une seconde thèse sur la fusion de l'information provenant de plusieurs sources : modèle surfacique 3D, infra rouge... dont le titre est : *Localisation d'objets urbains à partir de sources multiples*.

1. `http://www.developpement-durable.gouv.fr/IMG/pdf/F70_2012-05-30.pdf`

Troisième partie

Publications au cours de la thèse

Revue internationale

— [81] J. Pasquet, T. Desert, O. Bartoli, M. Chaumont, C. Delenne, G. Subsol, M. Derras, and N. Chahinian. Detection of manhole covers in high- resolution aerial images of urban areas by combining two methods. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(5) :1802–1807, May 2016

Conférence Internationale

— [79] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras. Speeding-up a convolutional neural network by connecting an SVM network. In IEEE International Conference on Image Processing, Phoenix, Arizona, USA, Sep 2016.

— [82] J. Pasquet, G. Subsol, M. Derras, and M. Chaumont. Optimizing color in- formation processing inside an SVM network. In IS&T International Symposium on Electronic Imaging, in Proceedings of Visual Information Processing and Communication VII, San Francisco, California, USA, Feb 2016.

— [85] L. Pibre, J. Pasquet, D. Ienco, and M. Chaumont. Deep learning is a good steganalysis tool when embedding key is reused for different images, even if there is a cover source-mismatch. In roceedings of Media Water- marking, Security, and Forensics, Part of IS&T International Symposium on Electronic Imaging, San Francisco, California, USA, FEB 2016.

— [78] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras. An efficient multi- resolution SVM network approach for object detection in aerial images. In MLSP'2015, The 25th IEEE International Workshop on Machine Learning for Signal Processing, Boston, USA, 2015.

— [80] J. Pasquet, T. Desert, O. Bartoli, M. Chaumont, C. Delenne, G. Subsol, M. Derras, and N. Chahinian. Detection of manhole covers in high-resolution aerial images of urban areas by combining two methods. In 2015 Joint Urban Remote Sensing Event (JURSE), pages 1–4, March 2015.

— J. Pasquet, S. Bringay and M. Chaumont, Steganalysis with cover-source mismatch and a small learning database, 2014 22nd European Signal Processing Conference (EUSIPCO), Lisbon, 2014, pp. 2425-2429.

— [50] M. Chaumont, L. Tribouillard, G. Subsol, F. Courtade, J. Pasquet and M. Derras, Automatic localization of tombs in aerial imagery : application to the digital archiving of cemetery heritage, Digital Heritage - International Congress 2013 - 28 Oct - 01 Nov, Marseille, France.

Conférence nationale

— [83] J. Pasquet, G. Subsol, and et M. Chaumont. Comparaison de la segmentation pixel et segmentation objet pour la d´etection d'objets multiples et variables dans des images. In COmpression et REpr´esentation des Signaux Audiovisuels, novembre 2014.

— J. Pasquet, S. Bringay, et M. Chaumont, Des millions d'images pour la stéganalyse : inutiles, CORESA'2013, COmpression et REprésentation des Signaux Audiovisuels, Le Creusot, France, 28-29 novembre, 2013, 6 pages

— [84] L. Pibre, M. Chaumont, D. Ienco, and J. Pasquet. Étude des réseaux de neurones sur la stéganalyse. In CORESA2016, COmpression et REprésentation des Signaux Audiovisuels, Nancy, France, May 2016.

Bibliographie

[1] D. Aldavert, A. Ramisa, R. López de Mántaras, and R. Toledo. Real-Time Object Segmentation using a Bag of Features Approach. In *13th International Conference of the ACIA*, L'Espluga de Francolí, Catalonia, Spain, 2010. IOS Press, IOS Press.

[2] D. Aldavert, A. Ramisa, R. Toledo, and R. López de Mántaras. Efficient Object Pixel-Level Categorization Using Bag of Features. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, Y. Kuno, J. Wang, J.-X. Wang, J. Wang, R. Pajarola, P. Lindstrom, A. Hinkenjann, M. L. E. ao, C. T. Silva, and D. Coming, editors, *Advances in Visual Computing*, volume 5875 of *Lecture Notes in Computer Science*, pages 44–54. Springer, 2009.

[3] D. Aldavert, A. Ramisa, R. Toledo, and R. Lopez de Mántaras. Fast and Robust Object Segmentation with the Integral Linear Classifier. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1046–1053, 2010.

[4] M. Anderson, R. Motta, S. Chandrasekar, and M. Stokes. Proposal for a standard default color space for the internet - srgb. In *4th Color and Imaging Conference, CIC 1996, Scottsdale, Arizona, USA, November 19-22, 1996*, pages 238–245, 1996.

[5] A. Angelova, A. Krizhevsky, and V. Vanhoucke. Pedestrian detection with a large-field-of-view deep network. In *Proceedings of ICRA 2015*, 2015.

[6] A. Angelova, A. Krizhevsky, V. Vanhoucke, A. Ogale, and D. Ferguson. Real-time pedestrian detection with deep network cascades. In *Proceedings of BMVC 2015*, 2015.

[7] R. Arandjelović and A. Zisserman. All about VLAD. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[8] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Spatio-Temporal Convolutional Sparse Auto-Encoder for Sequence Classification. In J. C. R. Bowden and K. Mikolajczyk, editors, *British Machine Vision Conference (BMVC)*, pages 124.1–124.12. BMVA Press, Sept. 2012.

[9] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110(3) :346–359, June 2008.

[10] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 :509–522, 2001.

[11] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910, June 2012.

[12] R. Benenson, M. Omran, J. Hosang, and B. Schiele. *Ten Years of Pedestrian Detection, What Have We Learned ?*, pages 613–627. Springer International Publishing, Cham, 2015.

[13] L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.

[14] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *CoRR*, abs/1508.00092, 2015.

[15] C.-C. Chang and C.-J. Lin. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2 :27 :1–27 :27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[16] M. Chaumont, L. Tribouillard, G. Subsol, F. Courtade, J. Pasquet, and M. Derras. Automatic localization of tombs in aerial imagery : Application to the digital archiving of cemetery heritage. In *Digital Heritage International Congress (DigitalHeritage), 2013*, volume 1, pages 657–660, Oct 2013.

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

[18] G. Ciocca, C. Cusano, and R. Schettini. Image orientation detection using LBP-based features and logistic regression. *Multimedia Tools and Applications*, 74(9) :3013–3034, 2015.

[19] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). 2016.

[20] F. Coelho and C. Ribeiro. Evaluation of global descriptors for multimedia retrieval in medical applications. In *2010 Workshops on Database and Expert Systems Applications*, pages 127–131, Aug 2010.

[21] D. Comaniciu and P. Meer. Mean shift : a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5) :603–619, May 2002.

[22] F. Courtade. Segmentation automatique d'images numeriques : Application a la detection des tombes dans un cimetiere. Technical report, Université Nice Sophia-Antipolis, 2012.

[23] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 207–212, New York, NY, USA, 1984. ACM.

[24] J. Dai, Y. Li, K. He, and J. Sun. R-FCN : object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

[25] Z. Daixian. SIFT algorithm analysis and optimization. In *2010 International Conference on Image Analysis and Signal Processing*, pages 415–419, April 2010.

[26] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893, June 2005.

[27] J. J. de Dios and N. Garcia. Face detection based on a new color space ycgcr. In *Image Processing, 2003. ICIP 2003*, volume 3, pages III–909–12 vol.2, Sept 2003.

[28] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet : A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255, June 2009.

[29] S. Dieleman, K. Willett, and J. Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notives of the royal astronomical society*, 450(2) :1441–1459, 2015.

[30] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection : A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311, June 2009.

[31] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf : A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, Reims, 2014.

[32] N. P. Doshi and G. Schaefer. Rotation-invariant local binary pattern texture classification. In *Electronics in Marine (ELMAR), 2012 Proceedings*, pages 71–74, Sept 2012.

[33] M. M. Dundar and J. Bi. Joint optimization of cascaded classifiers for computer aided detection. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[34] T. Durand, N. Thome, and M. Cord. Weldon : Weakly supervised learning of deep convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4743–4752, June 2016.

[35] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge : A retrospective. *International Journal of Computer Vision*, 111(1) :98–136, Jan. 2015.

[36] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR : A library for large linear classification. *Journal of Machine Learning Research*, 9 :1871–1874, 2008.

[37] D. A. Forsyth and J. Ponce. *Computer Vision : A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[38] J. Fournier, M. Cord, and S. Philipp-Foliguet. Retin : A content-based image indexing and retrieval system. *Pattern Analysis & Applications*, 4(2) :153–173, 2001.

[39] Y. Freund and R. E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119 – 139, 1997.

[40] Y. Freund and R. E. Schapire. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999.

[41] J. Gao and P.-N. Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 212–221, Washington, DC, USA, 2006. IEEE Computer Society.

[42] C. Garcia and M. Delakis. Convolutional Face Finder : A Neural Architecture for Fast and Robust Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(11) :1408–1423, Nov. 2004.

[43] T. Gevers, J. van de Weijer, and H. Stokman. *Color image processing : methods and applications : color feature detection : an overview*, chapter 9, pages 203–226. CRC press, 2006.

[44] R. Girshick. Fast R-CNN. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[45] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[46] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[47] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6) :610–621, Nov 1973.

[48] D. Harwood, T. Ojala, M. Pietikainen, S. Kelman, and L. Davis. Texture classification by center-symmetric auto-correlation, using kullback discrimination of distributions. *Pattern Recognition Letters*, 16(1) :1 – 10, 1995.

[49] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[50] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[51] B. Heisele, T. Serre, S. Prentice, and T. Poggio. Hierarchical classification and feature reduction for fast face detection with support vector machines. *Pattern Recognition*, 36 :2007–2017, 2003.

[52] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, July 2006.

[53] Y. Hwang, J. s. Kim, and I. Kweon. Determination of color space for accurate change detection. In *2006 International Conference on Image Processing*, pages 3021–3024, Oct 2006.

[54] S. Ioffe and C. Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 37 :448–456, 2015.

[55] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition ? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sept 2009.

[56] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision & Pattern Recognition*, pages 3304–3311, jun 2010.

[57] G. Jeon. Measuring and comparison of edge detectors in color spaces. In International Journal of Control and Automation, volume 6, pages 21–29, 2013.

[58] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1) :221–231, Jan. 2013.

[59] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe : Convolutional architecture for fast feature embedding. *arXiv preprint arXiv :1408.5093*, 2014.

[60] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1737–1743, 2016.

[61] Y. Ju, J. Guo, and S. Liu. A deep learning method combined sparse autoencoder with SVM. In *CyberC*, pages 257–260. IEEE Computer Society, 2015.

[62] C. Kertész and V. Oy. Texture-based foreground detection. *Image Processing and Pattern Recognition (IJSIP)*, 4(4), 2011.

[63] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. B. . Deep neural decision forests. June 2016.

[64] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[65] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[66] X. LOU, D. HUANG, L. FAN, and A. XU. An image classification algorithm based on bag of visual words and multi-kernel learning. *Journal of Multimedia*, 9(2), 2014.

[67] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2) :91–110.

[68] Y. Lu, L. Zhang, B. Wang, and J. Yang. Feature ensemble learning based on sparse autoencoders for image classification. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 1739–1745, July 2014.

[69] S. Lyu and E. P. Simoncelli. Nonlinear image representation using divisive normalization. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[70] W.-Y. Ma and B. Manjunath. Netra : A toolbox for navigating large image databases. *Multimedia Systems*, 7(3) :184–198, 1999.

[71] A. Marangoz, M. Oruc, S. Karakis, and H. Sahin. Comparison of pixel-based and object-oriented classification using ikonos imagery for automatic building extraction - safranbolu testfield. In *Fifth International Symposium Turkish-German Joint Geodetic Days*, pages 28–31, March 2006.

[72] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6) :555–559, 2003.

[73] B. McCane, K. Novins, and M. Albert. Optimizing cascade classifiers. 2005.

[74] W. S. McCulloch and W. Pitts. Neurocomputing : Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.

[75] F. Mindru, T. Tuytelaars, L. Van Gool, and T. Moons. Moment invariants for recognition under changing viewpoint and illumination. *Comput. Vis. Image Underst.*, 94(1-3) :3–27, Apr. 2004.

[76] F. Moosmann, E. Nowak, and F. Jurie. Randomized Clustering Forests for Image Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9) :1632–1646, Sept. 2008.

[77] Y. Mu, S. Yan, Y. Liu, T. Huang, and B. Zhou. Discriminative local binary patterns for human detection in personal album. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[78] S. Na, L. Xumin, and G. Yong. Research on k-means clustering algorithm : An improved k-means clustering algorithm. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 63–67, April 2010.

[79] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[80] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168, June 2006.

[81] P. Ott and M. Everingham. Implicit color segmentation features for pedestrian and object detection. In *2009 IEEE 12th International Conference on Computer Vision*, pages 723–730, Sept 2009.

[82] P. Over, G. Awad, M. Michel, J. Fiscus, W. Kraaij, A. F. Smeaton, G. Quèenot, and R. Ordelman. Trecvid 2015 – an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *Proceedings of TRECVID 2015*. NIST, USA, 2015.

[83] V. R. P. Ganesan and R. Rajkumar. Segmentation and edge detection of color images using CIELAB color space and edge detectors. In Emerging Trends in Robotics and Communication Technologies (INTERACT), 2010 International Conference on, pages 393–397, 2010.

[84] C. Pantofaru, C. Schmid, and M. Hebert. *Computer Vision – ECCV 2008 : 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III*, chapter Object Recognition by Integrating Multiple Image Segmentations, pages 481–494. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[85] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras. An efficient multiresolution SVM network approach for object detection in aerial images. In *MLSP'2015, The 25th IEEE International Workshop on Machine Learning for Signal Processing*, Boston, USA, 2015. IEEE.

[86] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras. Speeding-up a convolutional neural network by connecting an SVM network. In *IEEE International Conference on Image Processing*, Phoenix, Arizona, USA, Sep 2016. IEEE.

[87] J. Pasquet, T. Desert, O. Bartoli, M. Chaumont, C. Delenne, G. Subsol, M. Derras, and N. Chahinian. Detection of manhole covers in high-resolution aerial images of urban areas by combining two methods. In *2015 Joint Urban Remote Sensing Event (JURSE)*, pages 1–4, March 2015.

[88] J. Pasquet, T. Desert, O. Bartoli, M. Chaumont, C. Delenne, G. Subsol, M. Derras, and N. Chahinian. Detection of manhole covers in high-resolution aerial images of urban areas by combining two methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(5) :1802–1807, May 2016.

[89] J. Pasquet, G. Subsol, M. Derras, and M. Chaumont. Optimizing color information processing inside an SVM network. In *IS&T International Symposium on Electronic Imaging, in Proceedings of Visual Information Processing and Communication VII*, San Francisco, California, USA, Feb 2016. IEEE.

[90] J. Pasquet, G. Subsol, and et M. Chaumont. Comparaison de la segmentation pixel et segmentation objet pour la détection d'objets multiples et variables dans des images. In *COmpression et REprésentation des Signaux Audiovisuels*, novembre 2014.

[91] L. Pibre, M. Chaumont, D. Ienco, and J. Pasquet. Etude des réseaux de neurones sur la stéganalyse. In *CORESA2016, COmpression et REprésentation des Signaux Audiovisuels*, Nancy, France, May 2016.

[92] L. Pibre, J. Pasquet, D. Ienco, and M. Chaumont. Deep learning is a good steganalysis tool when embedding key is reused for different images, even if there is a cover source-mismatch. In *roceedings of Media Watermarking, Security, and Forensics, Part of IS&T International Symposium on Electronic Imaging*, San Francisco, California, USA, FEB 2016.

[93] P. H. O. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

[94] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once : Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[95] F. Rosenblatt. The perceptron – A perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[96] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[97] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing : Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[98] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks : Part III*, ICANN'10, pages 92–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[99] C. Schmid. Beyond bags of features : Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

[100] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat : Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[101] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008.*, pages 1–8, June 2008.

[102] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[103] J. Sivic and A. Zisserman. Video Google : a text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477 vol.2, Oct 2003.

[104] J. Sklansky. Image segmentation and feature extraction. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(4) :237–247, April 1978.

[105] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity : The all convolutional net. *International Conference on Learning Representations (ICLR)*, 2015.

[106] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1) :1929–1958, Jan. 2014.

[107] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[108] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013.

[109] H. Tamura, S. Mori, and T. Yamawaki. Textural features corresponding to visual perception. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(6) :460–473, June 1978.

[110] S. Tang and Y. Yuan. Object detection based on convolutional neural network. Report, Stanford University, 2015.

[111] Y. Tang. Deep learning using support vector machines. *CoRR*, abs/1306.0239, 2013.

[112] L. Tribouillard. Segmentation automatique d'images numeriques : Application a la detection des tombes dans un cimetiere. Technical report, Université Nice Sophia-Antipolis, 2013.

[113] K. van de Sande, T. Gevers, and C. Snoek. Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9) :1582–1596, Sept 2010.

[114] K. E. A. van de Sande, C. G. M. Snoek, and A. W. M. Smeulders. Fisher and VLAD with FLAIR. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[115] L. van der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9 :2579–2605, 2008.

[116] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[117] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367, June 2010.

[118] X. Wang, T. X. Han, and S. Yan. An HOG-LBP human detector with partial occlusion handling. In *2009 IEEE 12th International Conference on Computer Vision*, pages 32–39, Sept 2009.

[119] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep image : Scaling up image recognition. *CoRR*, abs/1501.02876, 2015.

[120] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5 :975–1005, Dec. 2004.

[121] Y. Xu, T. Mo, Q. Feng, P. Zhong, M. Lai, and E. I. C. Chang. Deep learning of feature representation with multiple instance learning for medical image analysis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1626–1630, May 2014.

[122] H. Zhu, X. Chen, W. Dai, K. Fu, Q. Ye, and J. Jiao. Orientation robust object detection in aerial images using deep convolutional neural network. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3735–3739, Sept 2015.

[123] Q. Zhu, S. Avidan, M. chen Yeh, and K. ting Cheng. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR06*, pages 1491–1498, 2006.