GéoPeuple

Rapport numéro	L3.1
Titre	**Vectorisations automatique de symboles**
Rédigé par	Jonathan Guyomard (LIP6)
État (final / en cours)	OK
Relu par	Matthieu Cord (LIP6)
Date	Octobre 2013

Table des matières

Table des figures

1

Introduction

Dans ce rapport, nous nous focalisons sur le problème de la vectorisation automatique dans le cadre du projet GéoPeuple. Cette tâche vise à détecter un ensemble de symboles, prédéfinis, qui ont été dessinés dans des cartes anciennes, dîtes cartes de Cassini, couvrant une grande partie du territoire français. A partir de différentes cartes numérisées (images), il s'agit pour le LIP6 de faire de la détection automatique d'objets (symboles) dans ces images. Plus précisément, nous proposons des approches issues de l'intelligence artificielle pour construire des machines logicielles capables de reconnaitre automatiquement la présence d'un symbole particulier dans une carte. On appelle ces machines des détecteurs. La problématique pour le LIP6 porte donc sur la construction des détecteurs les plus performants possibles sur ces cartes de Cassini.

Afin de réaliser un détecteur d'objets, il faut combiner séquentiellement une étape de traitement d'image et une étape de reconnaissance :

1. un algorithme d'extraction de descripteurs qui encodent l'information contenue dans l'image,
2. un algorithme de reconnaissance qui utilise les descripteurs calculés pour fournir une décision sur la présence d'un objet ou non.

L'objectif de ce travail, validé dans le cadre du projet, est donc double, bien que fort différent sur les 2 parties. En effet :

1. Sur la partie concernant le traitement d'images, notre objectif était de valider expérimentalement que des outils utilisés en vision par ordinateur dans le cadre de l'analyse de contenus photographiques d'images réelles, pouvaient s'appliquer également sur ce contexte de cartes anciennes dessinées. Comme nous allons le détailler, cette hypothèse a été largement validée dans ce projet. Un travail important afin d'évaluer les difficultés et les caractéristiques de différents outils a été mené. Beaucoup de facteurs entrent en jeu lors de cette étape qui ont des implications sur la suite : le processus de numérisation de la carte, les variations de couleurs ou de luminosité, les types d'objets, ou encore le contexte spatial local sur la carte. Ces difficultés ont déjà été abordées dans un précédent rapport.
2. La seconde partie traitant de la reconnaissance visait à produire un algorithme qui utilise les descripteurs calculés pour fournir une décision sur la présence d'un objet. Notre ambition affichée était d'utiliser des outils d'apprentissage statistique pour mener à bien cette étape. Nous avons testé des approches classiques, mais aussi proposé et validé des méthodes innovantes sur cette partie. Ce contexte d'apprentissage est largement développé dans ce rapport afin d'en préciser les enjeux et les caractéristiques.

1.1 Présentation générale de l'approche de vectorisation automatique du LIP6

Dans cette section, nous décrivons la méthode de manière générale, en détaillant une à une chaque étape de notre système afin d'obtenir en sortie un ensemble de détections (ou boîtes englobantes, ou *bounding box*) sur une image pour un objet recherché.

1.1.1 Apprentissage et détection, méthode de base

La manière la plus commune d'effectuer une détection sur des images est représentée sur le diagramme figure 1.

– **Annotations de données** : Toutes les méthodes d'apprentissage supervisée cherchent à produire des règles à partir de base d'apprentissage contenant des exemples. Ces exemples sont en général créés manuellement à partir de cas déjà traités et validés. L'annotation de données est une tâche importante en apprentissage, car trop peu ou de mauvaises annotations risquent de faire fortement chuter les performances du détecteur final.

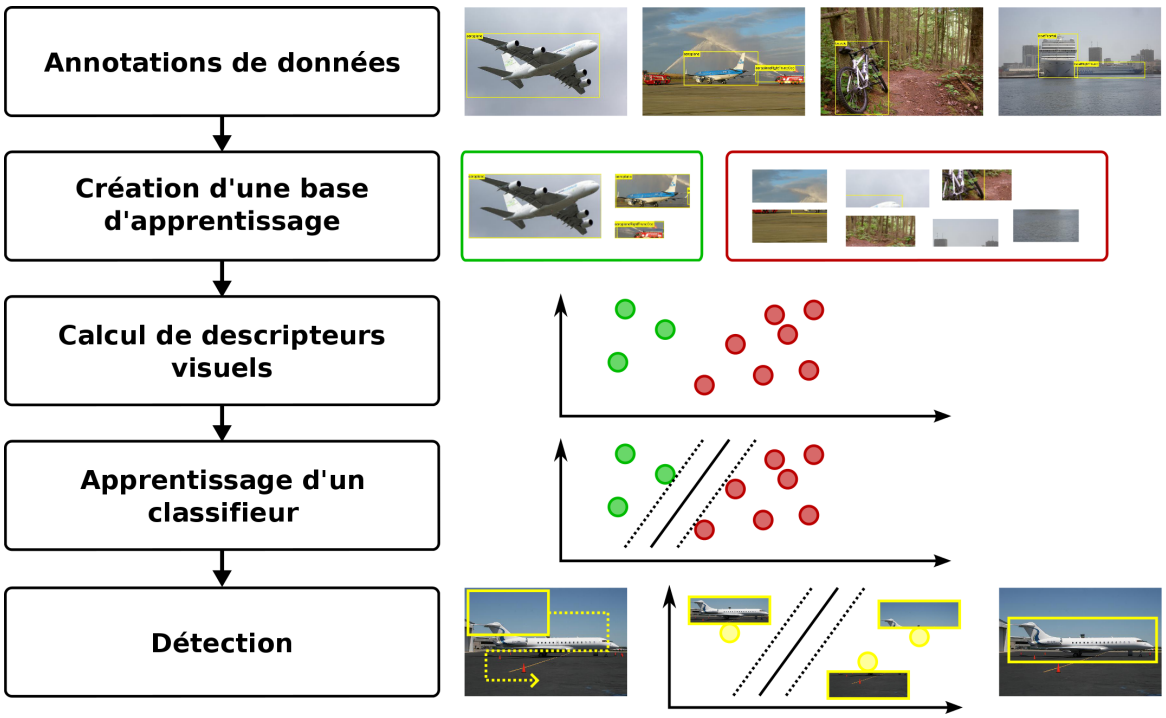


FIGURE 1 – Méthode de base pour la détection d'objets dans des images

– **Création d'une base d'apprentissage** : La création d'une base d'apprentissage pour un cas d'algorithme de détection peut se faire de plusieurs manières différentes, et c'est ici qu'une bonne annotation des données est importante. Pour le choix des exemples positifs, on choisira directement les annotations de notre base, tandis que pour les exemples négatifs, ceux-ci seront sélectionnés, soit :

  • exhaustivement, chaque fenêtre qui n'empiète pas sur une annotation est définie comme un exemple négatif. Dans ce cas, le nombre d'exemples négatifs peut rapidement augmenter et rendre l'apprentissage impossible en terme d'espace mémoire et de temps.

  • de manière aléatoire parmi l'ensemble des images annotées.

  • de manière intelligente, en sélectionnant uniquement les exemples négatifs vraiment informatifs.

– **Calcul de descripteurs visuels** : Chaque exemple de notre base d'apprentissage est représenté par une fenêtre dans une image. A partir de cette sous-image, nous calculons un descripteur visuel, comme décrit dans la section 2.1, qui nous permettra ensuite d'apprendre un classifieur à l'aide d'un algorithme d'apprentissage statistique supervisé.

– **Apprentissage d'un classifieur** : L'ensemble des descripteurs calculés précédemment permettent l'apprentissage d'un classifieur. Les différents algorithmes d'apprentissage supervisés sont détaillés dans la section 3.

– **Détection** : La dernière étape consiste à détecter les objets dans de nouvelles images. Chaque image est parcourue par un ensemble de fenêtres glissantes de différentes tailles et de différents rapport largeur/hauteur. Pour chacune de ces fenêtres, le même descripteur visuel utilisé pour la phase d'apprentissage est calculé, puis le classifieur attribue un score de confiance à ce descripteur, un score élevé indiquant une forte confiance pour la fenêtre de contenir l'objet recherché. La détection finale est alors l'ensemble des fenêtres ayant un score de confiance supérieur à une valeur fixée, en général zéro.

1.1.2 Ajout d'informations contextuelles inter-objets

Une thématique largement étudiée par la communauté de la vision par ordinateur est l'information contextuelle contenue dans une image. L'information contextuelle dans une image est, dans l'état de l'art, définie de plusieurs manières différentes : contexte région/région, contexte région/objet et contexte objet/objet. Chacune de ces méthodes sera présentée plus en détail dans la section 2.3. Étant donné la tâche du LIP6 pour le projet GéoPeuple, qui consiste à détecter des symboles dans d'anciennes cartes, il semble que le contexte objet/objet

soit le plus adapté pour ajouter de l'information et améliorer les performances de notre détecteur. Nous décrivons briévement comment cette information contextuelle est ajoutée à nos descripteurs dans le diagramme figure 2.

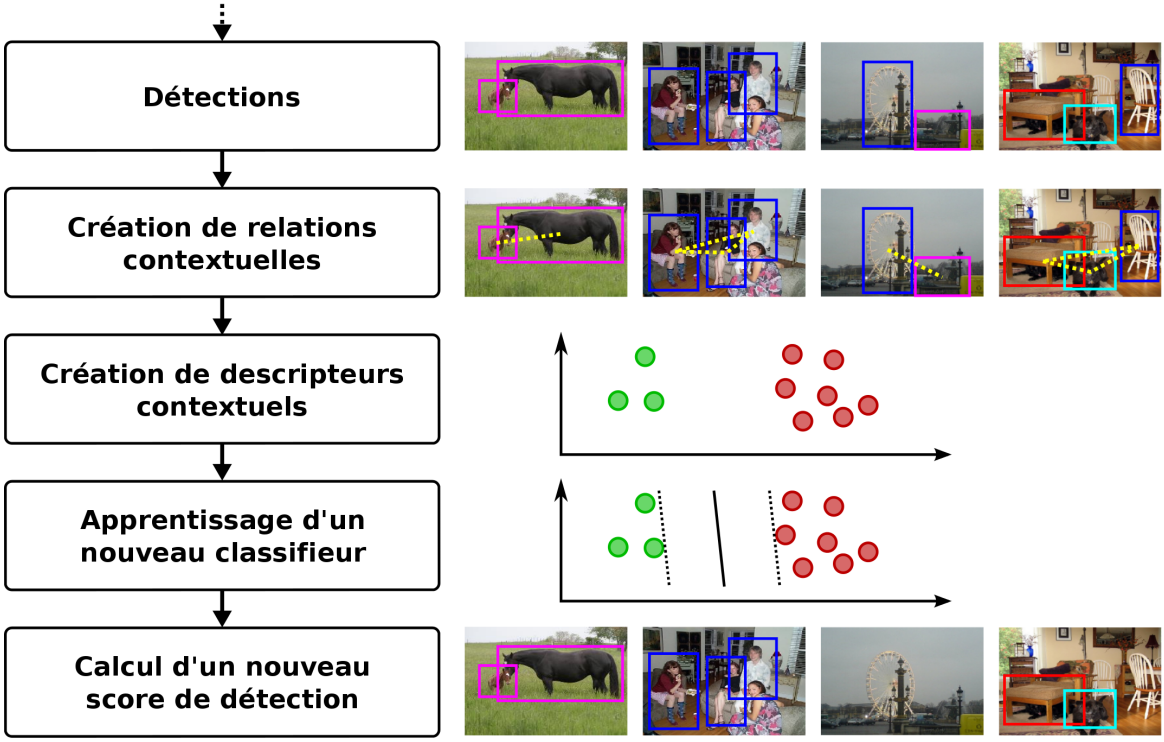


FIGURE 2 – Ajout d'informations contextuelles et suppression de fausses détections

– **Détections** : La première étape pour ajouter une information contextuelle objet/objet est d'effectuer une détection à l'aide de détecteurs préalablement entraînés pour chaque classe sur chaque image de la base d'apprentissage afin d'obtenir une liste de fenêtres scorées, qui sera la base de la création des relations et descripteurs entre objets.

– **Création de relations et descripteurs contextuelles** : L'étape fondamentale de l'ajout d'une information contextuelle est évidemment de décrire convenablement le contexte entre les objets. Nous décrivons plus en détail notre manière de procéder pour le représenter dans la section 4.

– **Apprentissage d'un nouveau classifieur** : Tout comme dans la section précédente, 1.1.1, les descripteurs construits permettent la création d'un nouveau classifieur avec une méthode d'apprentissage supervisée.

– **Calcul d'un nouveau score de détection** : Pour chaque fenêtre de la première étape de détection, le descripteur contextuel est calculé et noté par le nouveau classifieur. Si le contexte est suffisamment informatif, cette méthode permet le filtrage de nombreuses fausses détections.

1.1.3 Apprentissage actif

L'apprentissage actif est une forme d'apprentissage supervisé dans lequel l'algorithme d'apprentissage est capable d'interagir avec l'utilisateur afin d'obtenir de nouvelles données qui lui permettront de tendre vers une solution plus optimale. Dans certaines situations, et c'est le cas avec les cartes de Cassini, le nombre de données non annotées peut être élevé, et les annoter manuellement peut se révéler extrêmement long et fastidieux. Dans un tel cas, l'algorithme d'apprentissage peut activement faire appel à l'utilisateur afin de choisir les données qui lui serviront à s'entraîner. De cette manière, il arrive souvent que le nombre d'exemples requis pour apprendre un classifieur soit bien plus faible que dans un cas normal d'apprentissage supervisé. Le protocole est décrit sur le diagramme figure 3.

– **Sélection manuelle d'exemples incertains** : Après une détection sur les images de la base de test, l'algorithme propose à l'utilisateur un ensemble d'exemples qu'il devra annoter lui-même. En général ces exemples sont sélectionnés en fonction de leur proximité avec le seuil de détection du classifieur. En effet,

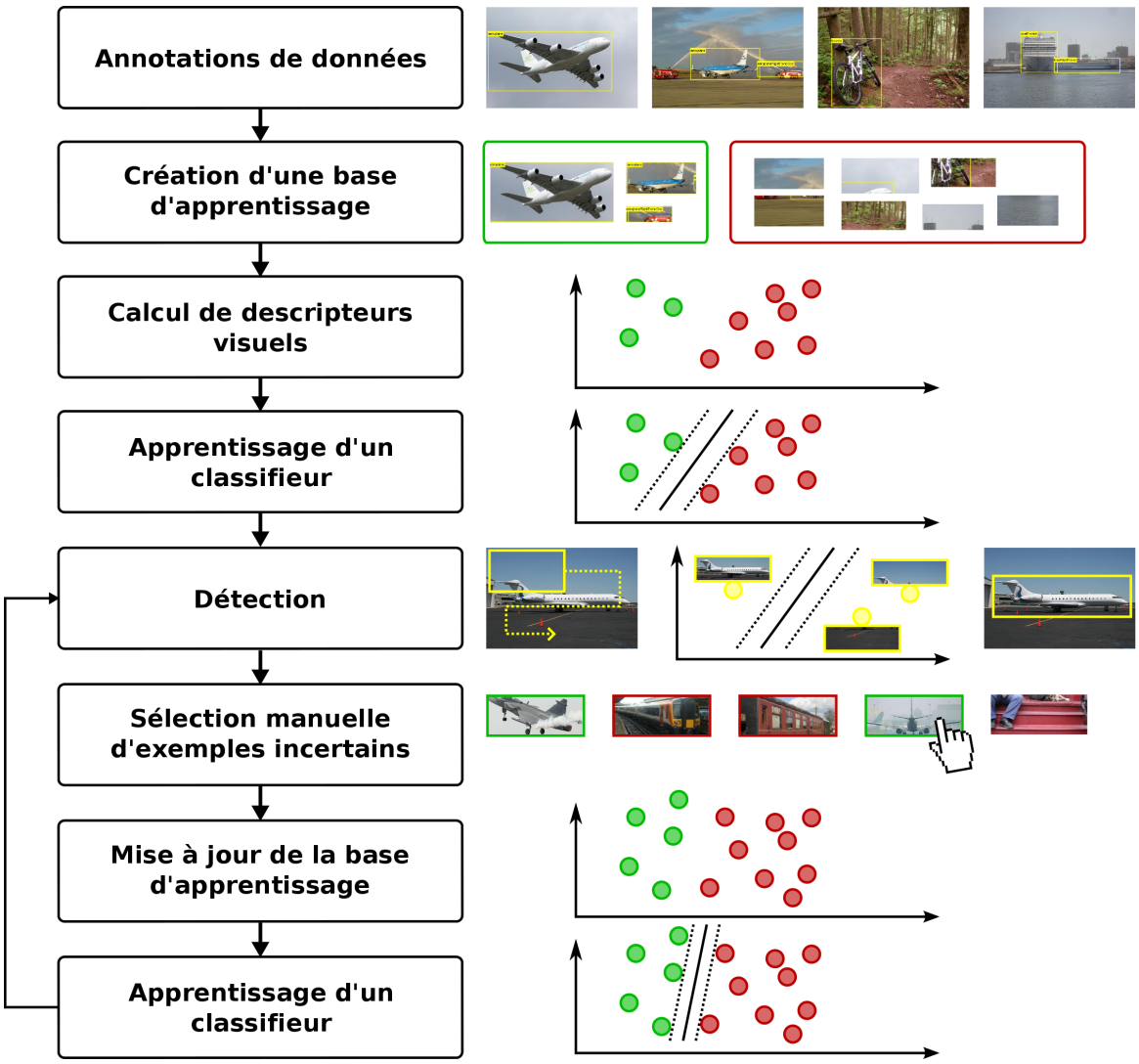


FIGURE 3 – Intéraction avec l'utilisateur permettant l'amélioration des performances des détecteurs

plus un exemple est proche du seuil de détection, plus il est considéré comme *incertain* et donc comme étant informatif pour la mise à jour du classifieur.

– **Mise à jour de la base d'apprentissage** : Les exemples annotés à l'étape précédente viennent enrichir la base de données d'apprentissage.

– **Apprentissage d'un nouveau classifieur** : Les deux étapes précédentes permettent l'apprentissage d'un nouveau classifieur qui permettra d'effectuer une nouvelle détection, normalement plus précise. Ces trois étapes peuvent être répété plusieurs fois afin d'améliorer continuellement la qualité de la détection.

1.2 Difficultés liées aux symboles à détecter

La principale difficulté dans la création d'un détecteur de symboles robuste pour les cartes de Cassini est le nombre particulièrement élevé de variations intra-classe, ou au contraire, de certaines ressemblances inter-classe. En particulier :

- les variations entre symboles d'une même classe. Les cartes ayant été tracées sur plusieurs décennies par différentes personnes, les symboles représentant une même classe sont parfois très variable en forme ou en taille (voir figure 4).

- les ressemblances entre symboles de classes différentes. Certains symboles représentant différentes classes sont parfois très similaires et donc difficilement différentiables, même pour un humain (voir figure 5).

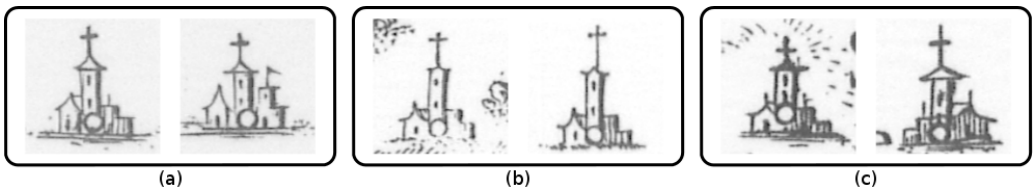


FIGURE 4 – Extrait d'un même type de symbole sur différentes zones de la carte de France, (a) Reims, (b) Grenoble, (c) Saint-Malo

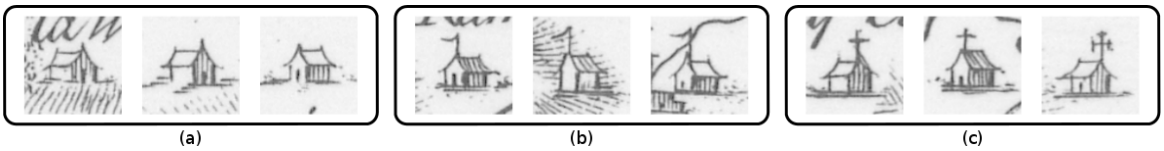


FIGURE 5 – Extrait de différentes classes représentées par des symboles ayant de fortes similarités, (a) écarts, (b) corps de garde, (c) chapelle et commanderie

- les variations de fond ou de contexte local. Selon que le symbole se trouve dans une zone de plaines, forestières, de montagne, près d'une rivière, d'une côte,...le fond de l'image peut énormément varier (voir figure 6).

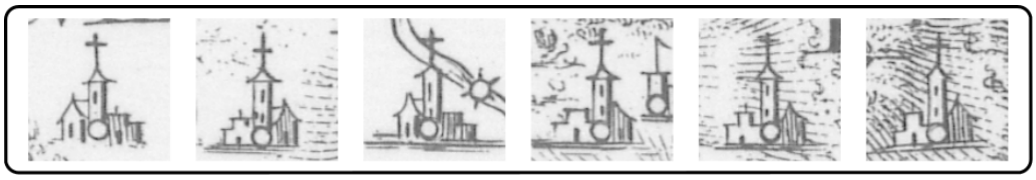


FIGURE 6 – Extrait de symboles d'une même classe avec de fortes variations dans le fond de l'image

Les figures 4, 5 et 6 représentent bien l'ensemble des variations et ambiguïtés visuelles auquel notre détecteur de symboles sera confronté. Il est donc fondamental de créer un détecteur qui sera robuste à ces différents changements dans les images.

2

Analyse d'images : les descripteurs utilisés pour le projet

2.1 Descripteurs d'image

L'extraction de descripteurs visuels est une étape cruciale dans tout processus d'analyse d'image, le but étant d'extraire certaines propriétés visuelles de régions d'une image via des opérations mathématiques au niveau des pixels. En fonction de la zone de l'image que l'on utilise pour extraire les descripteurs, on parle communément de descripteurs globaux ou de descripteurs locaux. Un descripteur global est calculé sur l'ensemble d'une image, reflétant les caractéristiques globales l'image, tandis qu'un descripteur local est calculé sur des régions relativement petites de l'image, permettant d'encoder les détails contenus dans ces zones. La plupart des approches d'extraction de descripteurs, qu'ils soient globaux ou locaux, se basent sur des propriétés visuelles universelles tel que la *couleur*, la *texture*, la *forme* ou les *contours*.

- **Couleur** : La perception de la couleur est très importante dans nombre d'applications, tel que l'imagerie digitale, les systèmes multimédia, la vision par ordinateur, le divertissement, ...C'est pourquoi elle est fortement utilisée pour créer des descripteurs visuels performants. L'article d'Alain Trémeau [12] fournit un aperçu de l'ensemble des méthodes utilisées, les tendances et les recherches futures sur la couleur dans le traitement des images et vidéos.
- **Texture** : La texture est un concept intuitif, car facilement perçu par l'être humain, mais qui manque d'une définition claire. En effet, la texture rassemble tout ce qui se rapporte à l'organisation spatiale des couleurs.
- **Forme** : La forme est surement la propriété visuelle de plus *haut niveau*, car étant une caractéristique extrêmement importante pour identifier et distinguer des objets. Les descripteurs de forme peuvent se regrouper dans deux types de méthodes : les méthodes basées sur les contours et les méthodes basées sur les régions. Une vue d'ensemble des descripteurs de forme existants se trouve dans l'article de Yang et al. [8].
- **Contours** : Les contours sont les points de l'image où il y a un fort changement de luminance (ou de niveau de gris) dans une direction privilégiée. Ils représentent en général les bords des objets et jouent donc un rôle très important dans les tâches de reconnaissance d'objet.

Nous détaillons à la suite le descripteur HOG [3] utilisé dans notre système qui exploite essentiellement une information de gradient agrégé sur un voisinage local.

2.2 Histogrammes de gradients orientés

Le but des Histogrammes de gradients orientés (HOG) [3] est de représenter l'apparence et la forme d'un objet dans une image grâce à la manière dont est réparti l'intensité du gradient ou la direction des contours. Ceci est effectué en divisant l'image en cellules et en calculant pour chaque cellule un histogramme des directions du gradient pour les pixels appartenant à cette cellule. La concaténation de ces histogrammes forme le descripteur.

- **Normalisation Gamma/couleur**
  L'extraction du descripteur a été évalué avec plusieurs représentations couleur des pixels : niveau de gris, RGB et LAB, avec en option une normalisation Gamma. Cette normalisation n'a qu'une faible incidence sur les performances et n'est donc pas obligatoire. Les espace de couleur RGB et LAB donnent des résultats comparables, alors que le niveau de gris réduit les performances. L'utilisation de l'information couleur est donc recommandée dans le cas où celle-ci est disponible.
- **Calcul des gradients**
  Les performances du détecteur sont sensibles à la manière dont ont été calculé les gradients, mais la technique la plus simple semble être la meilleure. Dalal et Triggs ont testé de pré-traiter l'image avec un

lissage Gaussien, puis de calculer les gradients en appliquant des filtres dérivatifs différents (plusieurs σ ont été testé pour le lissage Gaussien) :

Filtre dérivatif 1-D non centré	$\begin{bmatrix} -1 & 1 \end{bmatrix}$
Filtre dérivatif 1-D centré	$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$
Filtre dérivatif 1-D ajusté cubiquement	$\begin{bmatrix} 1 & -8 & 0 & 8 & -1 \end{bmatrix}$
Filtre de Sobel 3x3	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
Filtre diagonale 2x2	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ et $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Les meilleurs résultats sont obtenus en utilisant le filtré dérivatif centré, avec $\sigma = 0$ (aucun lissage). Pour les images couleurs, les gradients sont calculés séparément sur chaque canal couleur, le gradient ayant la norme la plus grande est gardé.

– **Calcul des histogrammes**
L'image est découpée en plusieurs cellules de petite taille et pour chaque cellule un histogramme est calculé. Chaque pixel d'une cellule vote pour une orientation entre 0° et 180° dans le cas non signé, ou entre 0° et 360° dans le cas signé. L'étape suivante est la normalisation des descripteurs, afin d'éviter les disparités dues aux variations d'illumination, ainsi que l'introduction de redondance dans le descripteur. Pour cela, les cellules sont regroupées par bloc (concaténation des histogrammes des cellules d'un bloc), le vecteur de valeur du bloc est ensuite normalisé. Les blocs se recouvrent, donc une même cellule peut participer plusieurs fois au descripteur final. Les normalisations possibles du descripteur final sont les suivantes (v représentant l'histogramme d'un bloc) :

L2-Norme	$v \rightarrow \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}$
L1-Norme	$v \rightarrow \frac{v}{\|v\|_1 + \epsilon}$
L1-Racine	$v \rightarrow \sqrt{\frac{v}{\|v\|_1 + \epsilon}}$

Ainsi qu'une quatrième norme, L2-Hys, qui consiste à calculer v par la L2-norme, limiter les valeurs maximales de v à 0.2, puis re-normaliser. Les normes L2-norme, L2-hys et L1-racine obtiennent des performances similaires, tandis que la L1-norme obtient de moins bons résultats, qui restent néanmoins meilleurs qu'une absence de normalisation.

2.3 Descripteurs contextuels

L'ensemble de la communauté de vision par ordinateur est d'accord pour dire que le contexte joue un rôle considérable dans la compréhension du contenu d'une image. En effet, de nombreuses études ont démontré l'importance du contexte dans la reconnaissance des objets par les humains, et quelques récentes méthodes de traitement d'image utilisant le contexte ont révéler une amélioration des performances vis à vis de méthodes plus basiques. Bien que le terme "*contexte*" soit souvent utilisé, il manquait de définition précise, et était vaguement assimilé à "*n'importe quelle information qui pourrait influencer la manière dont la scène et les objets s'entre-perçoivent*". Voici les différents types de contexte recensés par Santosh Divvala et al [11] :

– **Contexte local niveau pixel** : ce qui entoure la fenêtre de détection, le voisinage de l'image, la forme d'un objet
– **Contexte global d'une scène 2D** : statistiques sur l'ensemble de l'image
– **Contexte géométrique d'une scène 3D** : disposition de la scène, orientation des surfaces, occultations, points de context, ...
– **Contexte sémantique** : évènements ou activités représentés, catégorie de la scène, objets présents dans la scène et leurs relations spatiales, mots-clés
– **Contexte photogrammétrique** : hauteur et orientation de la prise de vue, distance de focale, distorsion de l'objectif
– **Illumination** : direction du soleil, couleur du ciel, contraste des ombres, ...

– **Contexte géographique** : localisation GPS, type de terrain, altitude, densité de population, ...
– **Contexte temporel** : suite d'images (si c'est une vidéo), proximité temporelle entre deux images, ...
– **Contexte culturel** : biais du photographe, sélection d'un ensemble d'images précis, ...

Dans la littérature, les méthodes utilisant le contexte pour la détection d'objets peuvent être classifiées en trois catégories, comme proposé dans l'article de Heitz et Koller [7] : le contexte *région-région*, *région-objet*, *objet-objet*. Nous n'utiliserons pas de méthode faisant appel à des *régions* (ou *segmentations*) de l'image car trop peu intéressante dans le cadre de la détection de symboles dans les cartes de Cassini, pour plus d'informations, se reporter aux articles de Heitz et Koller [7] et de Rabinovich et al [10].

2.3.1 Application aux cartes de Cassini

Tout comme une photographie peut contenir de l'information contextuelle, les cartes de Cassini renferment également de nombreuses informations autre que la forme des symboles que l'on cherche à détecter. Nous introduisons dans la section 2.3.1.1 un contexte *objet-objet* permettant de gérer une partie des ambiguïtés visuelles qu'il y a entre certains symboles, puis dans la section 2.3.1.2 un contexte plutôt orienté *région-objet*, qui contient une forte information a priori puisqu'il utilisera les données réelles des cartes de France actuelles pour améliorer les performances de notre détecteur.

2.3.1.1 Contexte inter-symboles

Dans tout système de détection automatique, il y aura inévitablement des faux positifs parmi les détections. En particulier dans les cartes de Cassini où apparaissent quelques cas évidents d'ambiguïtés visuelles, comme on peut le voir sur la figure 7. Pour pallier ce problème, nous proposons un nouveau descripteur contextuel basé sur une idée de l'article de Felzenswalb et al [5] que nous avons enrichi avec une information de position relative entre objets.

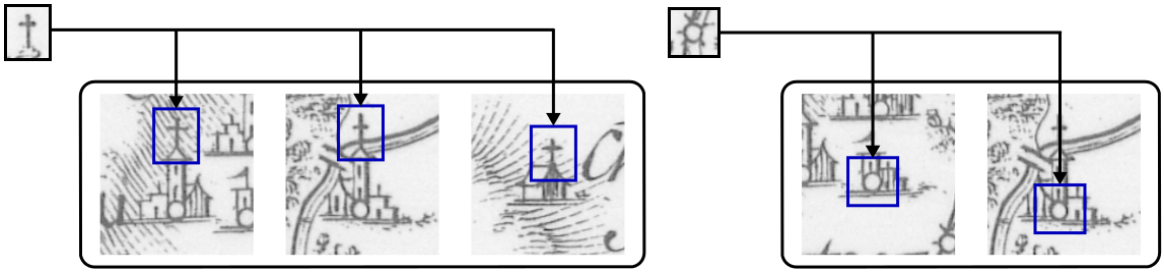


FIGURE 7 – Illustration d'ambiguïtés rencontrées entre différents symboles

On considère une zone de la carte où k détecteurs $(C_1, ..., C_k)$, basés uniquement sur l'apparence des symboles, sont appliqués. Ces détecteurs fournissent en sortie k ensembles de fenêtres $(D_1, ..., D_k)$, où chaque fenêtre $d \in D_i$ est définie par son centre $g = (g_x, g_y)$ et son score de classification s_i. Soit $w_j = (g_j, s_j), j \in [1, k]$ une fenêtre que l'on veut rescorer et d_i^* la fenêtre ayant le meilleur score de classification de l'ensemble D_i dans un périmètre d_{max} de g_w. Nous définissons le contexte d'une classe i pour la fenêtre w_j par :

$$n_i(w_j) = (\sigma(d_i^*), d_x, d_y)$$

où $\sigma(d_i^*)$ est défini par :

$$\sigma(d_i^*) = \begin{cases} d_i^*.s & \text{si } d_i^* \text{ existe} \\ -1 & \text{si il n'y a pas d'occurence de la classe } i \end{cases} \quad (1)$$

Pour le cas $i = j$, d_i^* ne doit pas se superposer avec w_j. d_x, d_y représentent la position relative de d_i^* par rapport à w_j, normalisés tel que $d_x, d_y \in [0, 1]$. Le contexte des k symboles pour la fenêtre w_j est alors :

$$\mathcal{N}(w_j) = (s_j, n_1(w), ..., n_k(w))$$

2.3.1.2 Contexte temporel

Une source d'information qui semble évidente lorsque l'on regarde ces cartes est l'information temporelle : "*si il existe une église à cet endroit au* $21^{ème}$ *siècle, il devait surement y avoir une église au même endroit au* $18^{ème}$ *siècle*". L'Institut National Géographique (IGN) possède ces informations et se propose de nous les fournir sous forme de masque binaire pour chaque carte que nous allons traiter, voir figure 8.

A partir de ces informations, on se propose d'extraire un nouveau descripteur contextuel. On considère une

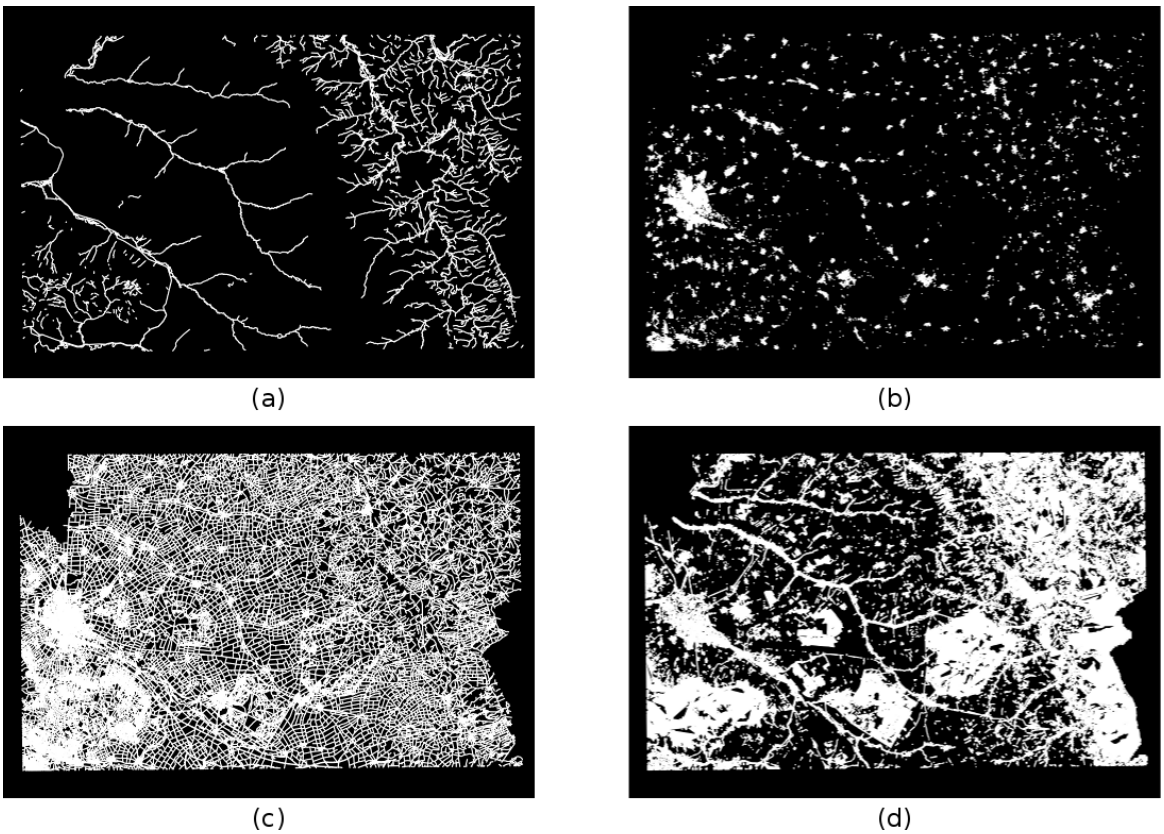


FIGURE 8 – Extraits de données actuelles sur la carte de Reims : (a) cours d'eau, (b) bâtis, (c) routes, (d) végétation

zone de la carte où un détecteur C, basé uniquement sur l'apparence a été appliqué. Ce détecteur fournit en sortie un ensemble de fenêtre D, où chaque fenêtre $d \in D$ est définie par son centre $g = (g_x, g_y)$ et son score de classification s. Soit k masques binaire $(M_1, ..., M_k)$ représentant différentes couches des données actuelles des cartes de France. Nous définissons le contexte d'une fenêtre w par :

$$RGE(w) = (p(M_1), ..., p(M_k))$$

où $p(M_i)$ corresponds à la présence d'un pixel blanc (valeur égale à 1) dans le masque M_i dans un périmètre d_{max} de g :

$$p(M_i) = \begin{cases} 1 & \text{si } M_i = 1 \text{ dans le périmètre } d_{max} \\ 0 & \text{sinon} \end{cases} \quad (2)$$

3

Algorithmes de classification supervisés

L'apprentissage supervisé est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage annotée. Les données d'apprentissage sont un ensemble d'*exemples* d'apprentissage. En apprentissage supervisé, chaque exemple est en fait une paire constituée d'un objet d'entrée, typiquement un vecteur de valeurs, et d'une sortie désirée. Un algorithme d'apprentissage supervisé va alors analyser les données d'apprentissage et générer une fonction qui permettra de prédire la valeur de sortie d'un nouvel exemple inconnu. Dans ce chapitre nous présentons brièvement le principe de fonctionnement des algorithmes d'apprentissage supervisé les plus populaire tel que le *boosting* et les *machines à vecteur de support*.

3.1 Boosting

Le boosting est un domaine de l'apprentissage automatique (ou *machine learning*). Le principe est de combiner un certain nombre de classifieurs, ou hypothèses, faibles (*weak classifier*) afin d'obtenir un classifieur final fort (*strong classifier*). Un classifieur faible doit être capable de reconnaître deux classes au moins aussi bien que le hasard ne le ferait, c'est-à-dire qu'il ne se trompe pas plus d'une fois sur deux en moyenne, si la distribution des classes est équilibrée. L'importance du poids d'un classifieur faible est donnée par la qualité de sa classification. A chaque itération, les poids des exemples sont mis à jour afin de donner plus d'influence à ceux qui étaient mal classés ce tour-ci. Il existe de nombreux algorithmes de *boosting*, mais ce qui les différencie est le plus souvent la manière de pondérer les exemples d'apprentissage et classifieurs faibles. L'algorithme le plus populaire étant l'*Adaptative Boosting* (ou *AdaBoost*) proposé par Robert Schapire et Yoav Freund [6].

Algorithm 1 AdaBoost

Entrées: Un ensemble d'apprentissage annoté $(x_1, y_1), \ldots, (x_m, y_m)$
où $x_i \in X$ sont les exemples et $y_i \in Y = \{-1, +1\}$ les annotations
Sorties: $H(x)$
Initialisation de la distribution des exemples $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$
pour $t = 1, \ldots, T$ **faire**
Trouver le classifieur $h_t : X \to \{-1, +1\}$ qui minimise l'erreur de classification ϵ_t
en fonction de la difficulté des exemples D_t
$\epsilon_t = \sum_{i=1}^{m} D_t(i)[y_i \neq h(x_i)]$
$h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{m} D_t(i)[y_i \neq h(x_i)]$
si $\epsilon_{min,t} < 0.5$ **alors**
h_t est sélectionné
sinon
L'algorithme s'arrête
fin si
Choix du poids du classifieur $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
Mise à jour de la pondération des exemples $D_{t+1}(i) = \frac{D_t(i)\, e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$
fin pour
$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

3.2 Machine à vecteurs de support

En apprentissage automatique, les machines à vecteurs de support (ou *SVM*) sont des algorithmes d'apprentissage supervisé utilisés pour traiter des cas de classification ou de régression. Étant donné un ensemble d'exemples d'apprentissage, chacun étiqueté comme appartenant à une classe parmi deux, l'apprentissage d'un SVM construit un modèle qui assignera les nouveaux exemples à l'une de ces deux classes. Les SVM ont été appliqués à de nombreux domaines, tel que la bio-informatique, la recherche d'information, la vision par ordinateur ou encore la finance, et selon les données, fournit des performances souvent de même ordre ou supérieures aux autres méthodes d'apprentissage supervisé.
Le problème d'un SVM linéaire peut s'écrire sous la forme suivante : soit $\mathcal{D}$ les données d'apprentissage, un

ensemble de n points de la forme $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathcal{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$ où y_i indique la classe à laquelle appartient le point $\mathbf{x}_i$. Chaque $\mathbf{x}_i$ est un vecteur de réel de taille p. Le but est de trouver l'hyperplan avec la marge maximum qui sépare les points ayant $y_i = 1$ de ceux ayant $y_i = -1$. Un hyperplan peut s'écrire sous la forme d'un ensemble de points $\mathbf{x}$ satisfaisant $\mathbf{w} \cdot \mathbf{x} - b = 0$, où $\cdot$ corresponds au produit scalaire et $\mathbf{w}$ au vecteur normal à l'hyperplan. Si les données d'apprentissage sont linéairement séparable, nous pouvons choisir deux hyperplans qui séparent les données de telle manière qu'aucun point ne soit entre les deux (voir figure 9.b), puis essayer de maximiser leur distance. Cette région entre les deux hyperplans est appelée *la marge* et leurs équations peuvent être écrites ainsi :

$$\mathbf{w} \cdot \mathbf{x} - b = 1 \text{ et } \mathbf{w} \cdot \mathbf{x} - b = -1$$

Avec un peu de géométrie, nous trouvons que la distance entre ces deux hyperplans est $\frac{2}{\|\mathbf{w}\|}$, donc nous voulons minimiser $\|\mathbf{w}\|$. Nous voulons également qu'aucun point ne tombe dans *la marge*, donc nous ajoutons les contraintes suivantes :

$$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1 \text{ pour les exemples avec } y_i = 1, \text{ et } \mathbf{w} \cdot \mathbf{x}_i - b \leq -1 \text{ pour les exemples avec } y_i = -1$$

Cela peut être réécrit sous la forme :

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \text{ pour tout } 1 \leq i \leq n.$$

Le problème d'optimisation est alors :

$$\text{Minimiser } \|\mathbf{w}\| \text{ sous la contrainte } y_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1, \text{ pour tout } 1 \leq i \leq n$$

En 1995, Corinna Cortes et Vladimir N. Vapnik [2] proposèrent une version modifiée des SVM à marge maximum en proposant un SVM à marge souple autorisant des exemples mal annotés. Si aucun hyperplan ne peut séparer les données, le SVM à marge souple choisira l'hyperplan séparant au mieux les deux ensembles.

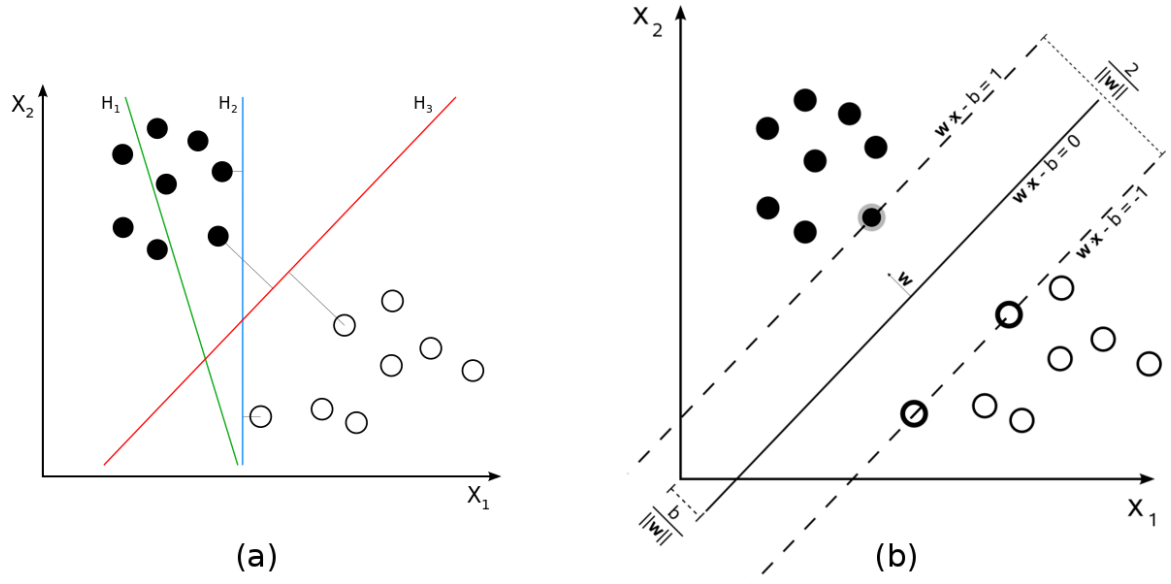


FIGURE 9 – (a) Différents hyperplans séparateurs : H_1 ne sépare pas les données correctement, H_2 sépare les données correctement avec une marge faible, H_3 sépare les données avec la marge maximum ; (b) Représentation d'une machine à vecteur de support

3.3 Descente de gradient stochastique pour l'apprentissage d'un SVM

Durant la dernière décennie, le nombre de données à augmenter plus vite que la vitesse de calcul des ordinateurs. Dans ce contexte, les algorithmes d'apprentissage automatique se sont vite retrouvés limités par la capacité de nos machines, et il en est de même dans le cadre du projet GéoPeuple, où la taille des cartes rends le nombre de données à traiter particulièrement élevé. Léon Bottou [1] a introduit une méthode de descente de gradient stochastique et a prouvé que cela était rapide et efficace dans le cas de l'optimisation d'un SVM linéaire.

Algorithm 2 Descente de gradient stochastique pour l'apprentissage d'un SVM linéaire

Entrées: labels, descripteurs, C, γ_0, γ_b, epochs
Sorties: w, biais

$w \leftarrow 0$
biais $\leftarrow 0$
$n \leftarrow$ nombre de descripteur
pour $it = 1$ à epochs **faire**
  Permuter aléatoirement les descripteurs et leur label
  $cpt = 0$
  pour $i = 1$ à n **faire**
    $\epsilon = \text{labels}(i) * (w * \text{features}(i)^T + \text{biais})$
    $t \leftarrow it * n + cpt$
    $\gamma_t \leftarrow \frac{\gamma_0}{(1+\frac{1}{C})*\gamma_0*t}$
    si $\epsilon \geq 1$ **alors**
      $w \leftarrow w * (1 - (\gamma_t * \frac{1}{C})))$
    sinon
      $w \leftarrow w * (1 - (\gamma_t * \frac{1}{C})) + \gamma_t * \text{labels}(i) * \text{features}(i)$
      biais $\leftarrow$ biais $+ \gamma_t * \text{labels}(i) * \gamma_b$
    fin si
    $cpt = cpt + 1$
  fin pour
fin pour

4

Expérimentations

Ce chapitre reprend les expérimentations qui ont été effectuées dans le cadre du projet GéoPeuple afin de déterminer les performances des détecteurs dans un cas réel. Les méthodes de mesure de performances utilisées dans cette section sont détaillées dans l'annexe B.

4.1 Données

L'ensemble des données utilisées pour effectuer nos expérimentations sont disponible sur le serveur FTP de l'*Institut Géographique National* :
Url : ftp2.ign.fr, **id :** anr_geoPeuple, **mot de passe :** non dispo (cf. tex) Les différentes classes et le nombre d'exemples pour chacune d'entre elle sont lister dans le tableau 10.

Hydro ponctuel			
	Reims	**Grenoble**	**Saint-Malo**
Source	190	126	10
Moulin, activité industrielle			
Eau	368	329	121
Forge	6	21	0
Mine.carriere	12	2	0
Vent.bois	94	1	94
Vent.pierre	0	0	42
Non religieux			
Château	102	195	90
Corps.de.garde	1	1	28
Ecart	161	1050	366
Gentilhommiere	16	109	112
Hameau	88	1768	1694
Relais.de.poste	6	2	0
Patrimoine.historique	2	0	1
Religieux			
Abbaye	8	9	3
Calvaire.cimetière	57	40	7
Chapelle.ou.oratoire	23	77	92
Eglise	287	348	139
Prieuré	5	5	3
Commanderie	3	0	0
Végétation ponctuelle			
Arbre.remarquable	9	0	1
Arbre.quelconque	19	34	2

FIGURE 10 – Tableau des données disponibles pour chaque carte annotée

Étant donné le nombre parfois restreint d'exemples pour certaines classes, nous nous sommes concentrés par la suite sur l'étude de huit classes en particulier : les moulins à eau (*Eau*), les moulins à vent (*Vent.bois*), les châteaux, les écarts, les hameaux, les calvaires (*Calvaire.cimetière*), les chapelles et les églises.

4.2 Histogrammes de gradients orientés

Nous présentons dans cette section les résultats obtenus en utilisant les histogrammes de gradients orientés décrit dans le chapitre 2.2. Nous avons choisi un descripteur qui découpe les fenêtre d'études en 4×4 cellules, et qui utilise des blocs de taille 2×2. L'orientation des gradients n'est pas signée et on échantillonne selon 9 directions. Le descripteur final est de taille 324. Le protocole de test est le suivant :

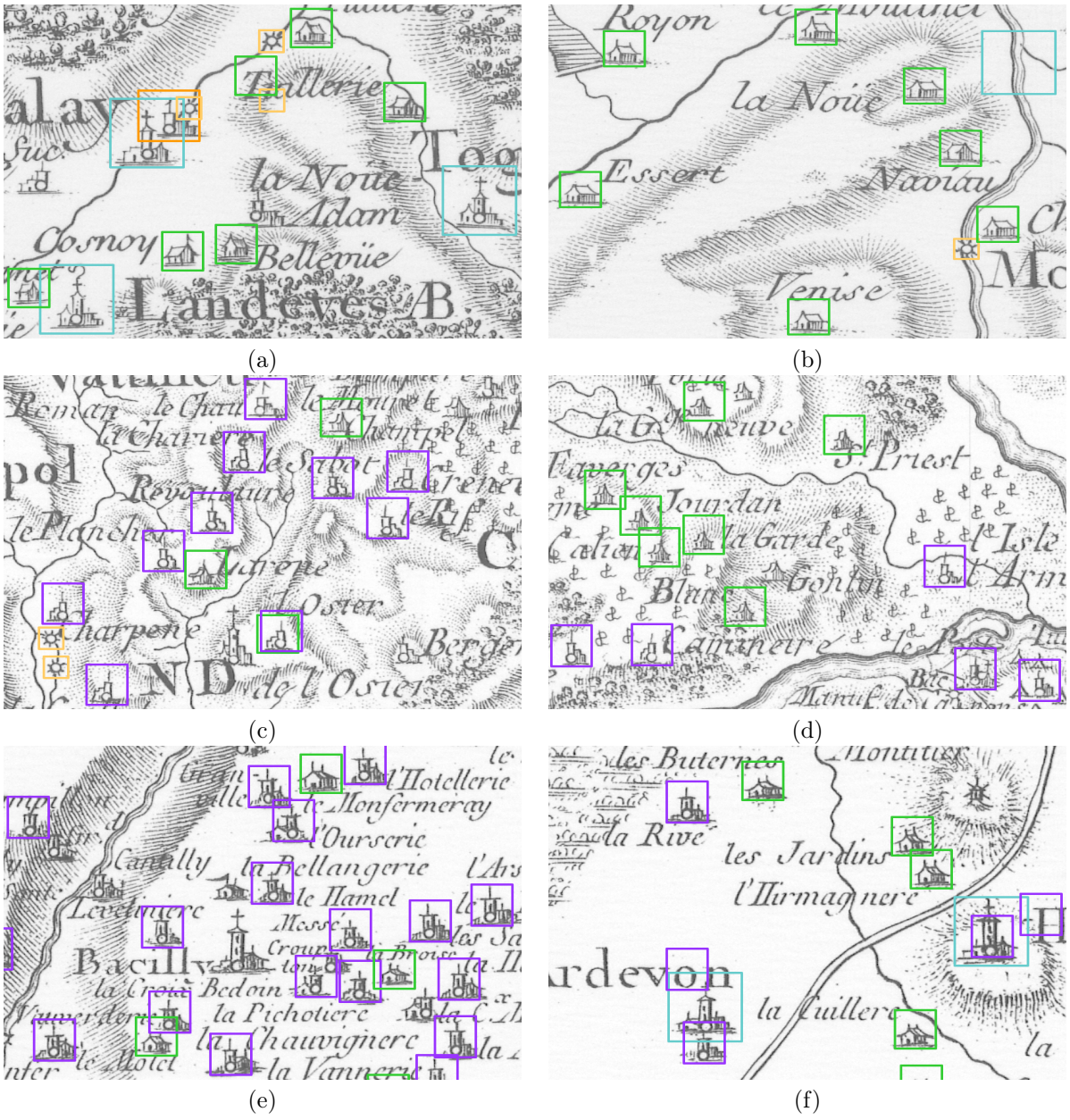
- Données utilisées : une carte (*Reims* ou *Grenoble*).
- Découpage de la carte en 9 parties distinctes.

– Apprentissage sur 8 parties puis évaluation sur la partie restante.

	Calvaire	Château	Moulin à eau	Ecart	Eglise	Hameau	Chapelle
MAP Reims	65.1	52.0	89.8	84.1	78.2	74.6	38.5
MAP Grenoble	26.4	30.7	91.8	80.4	68.9	89.7	25.7

FIGURE 11 – Performances en précision moyenne des HOG sur *Reims* et *Grenoble*

Les résultats en terme de précision moyenne sont présentés dans le tableau 11. Ces chiffres représentent déjà d'excellents résultats en détection, comme on peut le voir sur les figures 12.



(a) (b) (c) (d) (e) (f)

FIGURE 12 – Résultats de détections effectuées sur les cartes de (a-b) *Reims*, (c-d) *Grenoble* et (e-f) *Saint-Malo*

4.3 Contexte inter-objets

Nous présentons dans cette section les résultats obtenus en utilisant le descripteur contextuel décrit dans le chapitre 2.3.1.1. Les expérimentations pour évaluer ce descripteur furent particulièrement longues, en effet, une détection doit être effectuée pour chaque symbole avant de pouvoir extraire le descripteur contextuel d'une fenêtre. De plus, de nombreuses questions furent posées pour connaître la manière de procéder pour évaluer les performances de ce descripteur. Nous nous sommes arrêté sur le protocole de test suivant :

– Données utilisées : une carte (*Reims* ou *Grenoble*).

– Découpage de la carte en 9 parties distinctes.
– Apprentissage sur 8 parties d'un classifieur visuel pour chaque symbole.
– Détection sur les 9 parties pour chaque symbole.
– Les fenêtres les mieux classées sont gardées pour former le contexte.
– Calcul des descripteurs contextuels pour les fenêtres ayant le plus fort score visuel (les *hard negatives*).
– Apprentissage sur 8 parties d'un classifieur contextuel puis évaluation sur la partie restante.
– Test de Student pour évaluer si le gain est significatif.

	HOG	CON	GAIN
Calvaire	65.1	68.9	+3.8
Château	52.0	52.1	+0.1
Moulin à eau	89.8	90.6	**+0.7**
Ecart	84.1	84.7	+0.6
Eglise	78.2	78.0	-0.2
Hameau	74.6	75.8	+1.2
Chapelle	38.5	38.0	-0.5

(a)

	HOG	CON	GAIN
Calvaire	26.4	28.6	+1.2
Château	30.7	27.1	**-3.6**
Moulin à eau	91.8	93.2	**+1.4**
Ecart	80.4	80.9	+0.5
Eglise	68.9	69.1	+0.2
Hameau	89.7	90.1	**+0.4**
Chapelle	25.7	23.8	-1.9

(b)

FIGURE 13 – Résultats de l'intégration du contexte inter-symboles, (a) Reims, (b) Grenoble

Les résultats en terme de précision moyenne sont présentés dans le tableau 13. Les gains écrit en **gras** indiquent des gains significatifs selon le test de Student. La figure 14 représente un vecteur de poids après apprentissage d'un classifieur avec ce descripteur contextuel, et la figure 15 représente les vecteurs de poids appris pour chaque classe séparément, ce qui permet d'étudier l'influence des liens entre chacune d'entre elles. Comme on

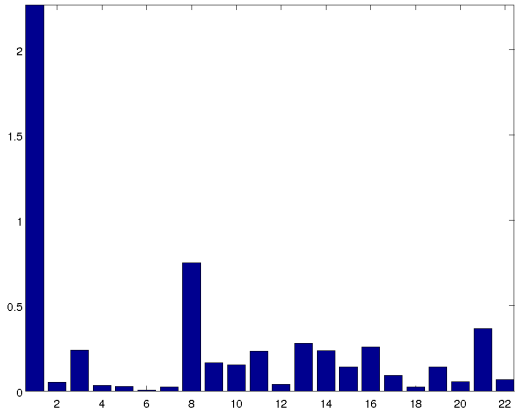
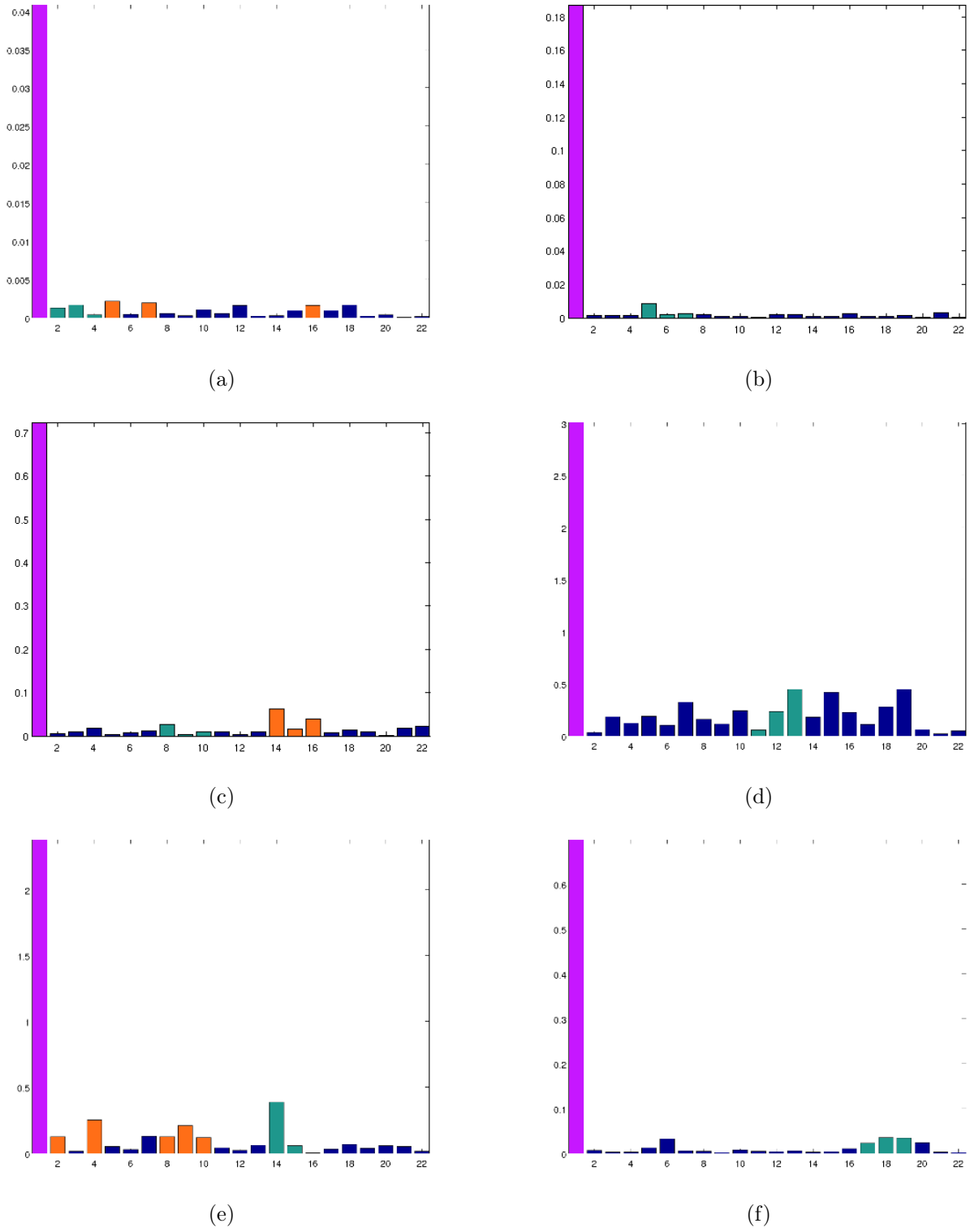


FIGURE 14 – Composition du vecteur : (1) influence du HOG, (2-4) influence des calvaires, (5-7) influence des châteaux, (8-10) influence des moulins à eau, (11-13) influence des écarts, (14-16) influence des églises, (17-19) influence des hameaux, (20-22) influence des moulins à vent.

peut le voir sur la figure 15, l'influence du contexte pour réévaluer le score d'une fenêtre s'avère minime. En effet, après cross-validation sur 9 parties de chaque carte, les tendances redondantes dans les vecteurs de poids sont représentées en orange. On remarque alors que le poids des liens entre classes est extrêmement faible par rapport au poids de la représentation visuelle du symbole. De plus, seule quelques relations sont réellement confirmées par leur redondance entre cartes. Finalement, le contexte inter-symboles ne semblent pas améliorer drastiquement les performances, en apportant uniquement un gain significatif sur la classe *moulin à eau* après validation croisée sur *Reims* et *Grenoble*.

4.4 Contexte RGE

Nous présentons dans cette section les résultats obtenus en utilisant le descripteur contextuel décrit dans le chapitre 2.3.1.2. Les données de la *BD TOPO* étant nombreuses, nous avons choisi celles paraissant les plus pertinentes pour extraire de l'information entre deux époques : bâti indifférencié, bâti industriel, bâti remarquable, chemin, cimetière, religieux, zone habitation, point d'eau, route, cours d'eau et végétation. L'ensemble des données disponibles est visible dans le tableau 16. Le vecteur de description d'une fenêtre de l'image sera donc de dimension 12, une dimension pour le score visuel (HOG) et onze dimensions pour le contexte représenté



(a) (b) (c) (d) (e) (f)

FIGURE 15 – Vecteur de poids du contexte inter-symboles par classe. La première colonne, en violet, indique l'influence du score HOG, les colonnes en orange indiquent des valeurs *influentes*, les colonnes en turquoise indiquent l'influence d'un symbole par rapport à lui même. (a) calvaires, (b) châteaux, (c) moulins à eau, (d) écart, (e) église, (f) hameaux.

par les masques RGE.
Le protocole de test est identique à celui utilisé pour évaluer les performances du contexte inter-objets :

- Données utilisées : une carte (*Reims* ou *Grenoble*).
- Découpage de la carte en 9 parties distinctes.
- Apprentissage sur 8 parties puis évaluation sur la partie restante.
- Test de Student pour évaluer si le gain est significatif.

Réseau routier			
Chemin	Route nommée	Route primaire	Route secondaire
Route	Surface route	Toponyme communication	
Voies ferrées et autres			
Aire triage	Gare	Toponyme ferré	Tronçon voie ferrée
Transport énergie			
Conduite	Ligne électrique	Poste transformation	
Hydrographie			
Canalisation eau	Hydronyme	Point eau	Réservoir eau
Surface eau	Tronçon cours eau		
Bâti			
Bâti indifférencié	Bâti industriel	Bâti remarquable	Cimetière
Construction linéaire	Construction surfacique	Piste aérodrome	Réservoir
Terrain sport			
Végétation			
Zone végétation			
Orographie			
Ligne orographique	Oronyme		
Administratif			
Chef lieu	Commune		
Zone activité			
Administratif militaire	Culture loisirs	Espace naturel	Gestion eaux
Hydrographie	Industriel commercial	Orographie	Religieux
Santé	Science enseignement	Sport	Transport
Zone habitation	Surface activité		
Toponymes			
Lieu dit habité	Lieu dit non habité	Toponyme divers	

FIGURE 16 – Tableau des données BD TOPO disponibles

Les résultats en terme de précision moyenne sont présentés dans le tableau 17. Les gains écrit en **gras** indiquent des gains significatifs selon le test de Student. Comme on peut le constater, les gains sont bien plus important que ceux enregistrés avec le contexte inter-objets. De plus, lorsque l'on observe les valeurs apprises par notre classifieur, les interactions entre les objets et leur contexte sont nettement plus marquées, voir figure 18. En effet, pour la détection des églises, les masques les plus informatifs (représentés par les colonnes orange dans les histogrammes de la figure 18) sont ceux liés aux bâtis, au religieux et au zone d'habitation, tandis que pour la détection des moulins à eau la présence d'un cours d'eau apportera plus de poids lors de la prédiction finale.

	HOG	RGE	GAIN
Calvaire	65.1	64.5	-0.5
Château	52.0	54.7	**+2.7**
Moulin à eau	89.8	91.1	**+1.3**
Ecart	84.1	84.5	+0.4
Eglise	78.2	81.4	**+3.2**
Hameau	74.6	75.2	+0.6
Chapelle	38.5	36.2	-2.3
MAP	68.9	69.7	+0.7

(a)

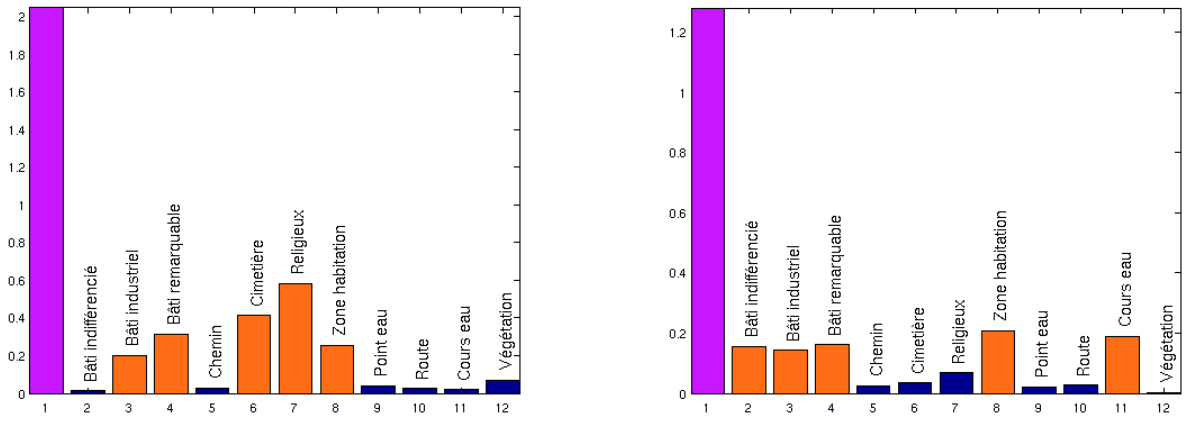
	HOG	RGE	GAIN
Calvaire	26.4	26.4	0.0
Château	30.7	34.9	**+4.2**
Moulin à eau	91.8	92.6	**+0.8**
Ecart	80.4	80.7	+0.3
Eglise	68.9	72.4	**+3.5**
Hameau	89.7	90.0	**+0.3**
Chapelle	25.7	27.1	+1.4
MAP	59.1	60.6	**+1.5**

(b)

FIGURE 17 – Résultats de l'intégration du contexte RGE, (a) Reims, (b) Grenoble

4.5 Apprentissage actif

Comme présenté dans la section 1.1.3, l'apprentissage actif consiste à faire appel à l'utilisateur pour que celui-ci annote manuellement certains exemples incertains de manière à améliorer les performances d'un détecteur. Le protocole de test utilisé est le suivant :



(a) (b)

FIGURE 18 – Vecteur de poids après apprentissage du contexte RGE, (a) église, (b) moulin à eau. La première colonne indique le poids du score visuel (HOG), les colonnes suivantes indiquent le poids de chaque masque RGE pour la détection des symboles.

- Données d'apprentissage : une carte entière (*Reims* ou *Grenoble*).
- Données d'évaluation : une carte entière (*Reims* ou *Grenoble* selon celle choisie en apprentissage).
- Un classifieur est appris sur les données d'apprentissage.
- Une première détection est effectuée sur les données d'évaluation et les fenêtres les mieux classifiées sont conservées.
- Un sous-ensemble des fenêtres conservées est présenté à l'utilisateur pour qu'il les annote manuellement.
- Le classifieur est mis à jour en fonction des annotations de l'utilisateur.
- L'étape d'annotations est répétée jusqu'à ce que la détection convienne à l'utilisateur.

La figure 19 montre l'évolution du rappel (en fixant un nombre de faux positifs à 2500) et de la précision moyenne d'une détection en 6 itérations pour la classe *écart*. À chaque itération l'utilisateur peut choisir d'annoter une dizaine d'exemples. L'apprentissage a été effectué sur la carte de *Reims* tandis que la détection est réalisée sur la carte de *Grenoble*, qui contient 1050 exemples positifs. Comme on peut le voir, en très peu d'itérations les résultats s'améliorent fortement avec une précision moyenne passant de 61.3% à 83.1% et un rappel (en admettant que l'utilisateur ne filtre que 2500 exemples négatifs) passant de 79.1% (830 exemples) à 94.2% (989 exemples).

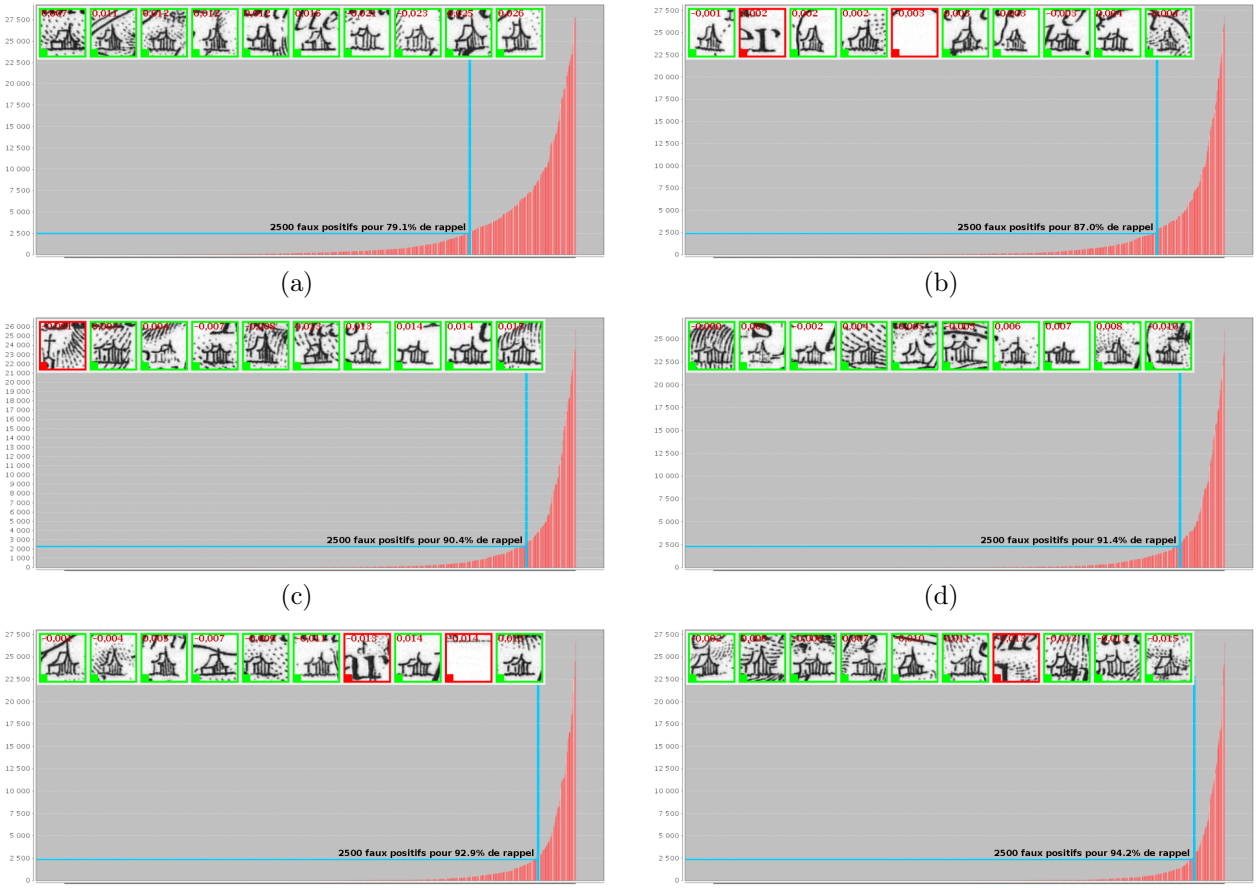


FIGURE 19 – Progression des performances de détection durant 6 itérations d'apprentissage actif. L'utilisateur choisi de ne filtrer manuellement que 2500 détections, nous observons l'évolution du rappel et de la précision moyenne après chaque itération : (a) $R = 79.1\%, AP = 61.3\%$, (b) $R = 87.0\%, AP = 71.8\%$, (c) $R = 90.4\%, AP = 77.2\%$, (d) $R = 91.4\%, AP = 78.7\%$, (e) $R = 92.9\%, AP = 80.7\%$, (f) $R = 94.2\%, AP = 83.1\%$

A

Base de données

Chaque nouvelle méthode de classification en apprentissage supervisé requiert d'être correctement testée et validée sur des données appropriées afin d'évaluer si celle-ci est utilisable pour des applications bien réelles. Il est important que ces données soient choisies de manière à être représentatives pour l'application que l'on cherche à créer. Une base de données d'apprentissage est un ensemble de couples (x, y), où x est un exemple et y son label. Cet ensemble permettra par la suite de produire automatiquement une *fonction de prédiction* f à laquelle on présentera de nouveaux exemples x auxquels on associera une sortie $f(x)$ qui nous servira à définir leur label.

Dans le cadre du projet *GéoPeuple*, nous avons créé une application de vectorisation automatique de symboles dans des cartes de France du 18ème siècle. La méthode utilisée peut être également appliquée pour la détection d'objets plus génériques, c'est pourquoi nous présentons dans la suite de cette annexe deux bases de données totalement différentes qui nous ont permis de tester la fiabilité et la flexibilité de notre méthode de détection.

A.1 Carte de Cassini

La Carte de Cassini ou Carte de l'Académie est constituée d'une collection de 180 feuilles, 154 feuilles au format 104 cm x 73 cm et 26 feuilles à des formats divers. Elles ont été gravées sur des plaques de cuivre conservées par l'Institut géographique national (IGN) qui diffuse les sorties papier en noir et blanc. Trois zones, a la topologie différente, ont été choisi pour être étudier : Saint-Malo pour sa côte maritime, Reims pour le paysage de plaine et Grenoble pour la haute montagne.

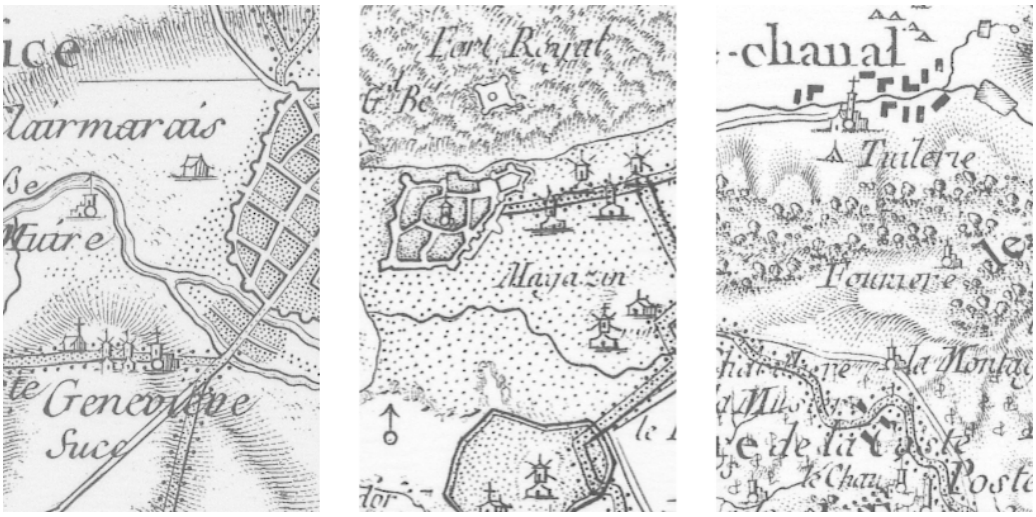


FIGURE 20 – Extraits de différentes cartes de Cassini

Les rapports L2.0-2 "Présentation des données" et L2.1-2 "Contenu de la carte de Cassini et sa modélisation en vue de sa vectorisation" reprennent en détail le contenu et la méthode d'annotation des données des cartes de Cassini.

A.2 Pascal VOC 2007

La base de données "PASCAL Visual Object Classes (VOC)" [4] est une référence dans la communauté de la reconnaissance et détection d'objets dans les images en fournissant une base publique de photographies et de leurs annotations. Cette base de données a été acceptée comme étant la référence pour l'évaluation d'algorithme de détection d'objets dans des images naturelles. La base de données PASCAL VOC 2007 est en fait un ensemble de photos d'utilisateur de Flickr, un site de partage de photos en ligne. Elle contient les annotations de 20 classes

différentes : personne, oiseau, chat, vache, chien, cheval, mouton, avion, vélo, bateau, bus, voiture, moto, train, bouteille, chaise, table, pot de fleur, sofa et écran. Il y a au total 9963 images séparées en trois sous-ensembles : apprentissage (2501 images), validation (2510 images) et test (4952 images). Cet ensemble d'images contient un nombre significatif de variations dans la taille, la position, l'illumination ou la position des objets, rendant la tâche de détection extrêmement complexe.

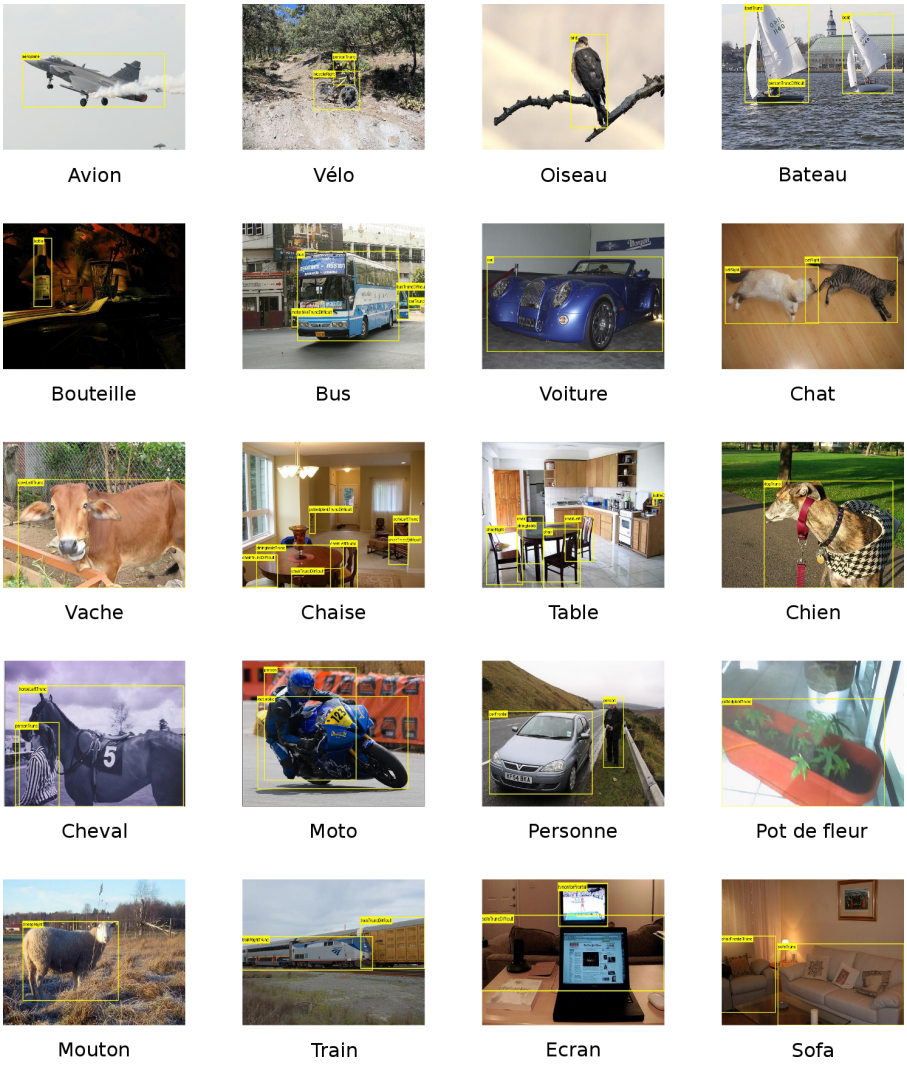


FIGURE 21 – Extrait de la base de données PASCAL VOC 2007

B

Méthode d'évaluation

Afin d'effectuer des comparaisons entre nos différentes méthodes de vectorisation, ou de simplement quantifier les performances de nos détecteurs, il est important de définir des méthodes d'évaluation robustes. De manière classique en apprentissage supervisé, l'évaluation des performances d'un système peut se faire de deux façons bien distinctes. Soit une évaluation en classification, où chaque exemple de la base de données est pris en considération, soit une évaluation en détection, où l'on considèrera plutôt la localisation des fenêtres correctement détectées. Dans les deux cas, les détecteurs offrent en sortie un score de confiance pour chaque position, où une grande valeur indique une forte confiance de présence de l'objet dans la fenêtre.

B.1 Taux d'erreurs de classification

Le taux d'erreurs de classification est une méthode simple pour évaluer les performances d'une fonction de classement. Soit une base de données contenant X exemples appartenant à la classe recherchée et Y exemples n'y appartenant pas. Nous confrontons les prédictions de notre fonction de classification avec les vraies valeurs de la variable à prédire, et il en résulte alors une matrice de confusion avec en ligne la vraie classe d'appartenance et en colonne la classe d'appartenance prédite. Le taux d'erreurs de classification est simplement le nombre de mauvais classement, c'est à dire lorsque la prédiction ne coïncide pas avec la vraie valeur. Le taux d'erreurs de classification est facile à interpréter et mettre en oeuvre, mais il n'est pas représentatif dans le cas d'une base de données à fort déséquilibre entre classe et il est fortement dépendant de seuil de détection de notre classifieur. Par exemple, pour une base de données avec $X = 100$ et $Y = 1000000$, si le classifieur entraîné estime que tous les exemples n'appartiennent pas à la classe recherchée, le taux d'erreur sera de $T = \frac{X}{X+Y} = \frac{100}{1000100} = 0.01\%$, ce qui n'est évidemment pas représentatif des performances de la fonction de classification.

B.2 Courbes de Rappel-Précision

En classification, les critères de mesure de performances les plus couramment utilisés sont le rappel et la précision. La précision est la proportion d'exemples pertinents retournés sur l'ensemble des exemples retournés, tandis que le rappel est la proportion d'exemples pertinents retournés sur l'ensemble des exemples pertinents de la base. Si l'on reprends l'exemple précédent avec $X = 100$ exemples appartenant à la classe recherchée et $Y = 1000000$ exemples n'y appartenant pas. Si le classifieur retourne environ 200 exemples contenant 50 exemples pertinents, la précision sera $p = \frac{50}{200} = 25\%$ et le rappel sera $r = \frac{50}{100} = 50\%$. On remarque alors que les deux mesures prises séparément ne sont pas représentatives. En effet si l'on paramètre le classifieur pour ne retourner qu'un seul exemple dont on est sur de sa pertinence, la précision sera $p = \frac{1}{1} = 100\%$ et le rappel sera $r = \frac{1}{100} = 1\%$. Au contraire si le classifieur est paramétré pour être très souple et retourner tous les exemples, la précision sera $p = \frac{100}{1000100} = 0.01\%$ et le rappel sera $r = \frac{100}{100} = 100\%$.

Les performances d'un classifieur ne se réduisent donc pas à un bon score en précision ou en rappel. Le système retournant un ensemble de données triées, il est important de considérer l'ordre dans lequel sont renvoyées les réponses. En calculant la précision et le rappel pour chaque réponse renvoyée, il est possible de tracer la courbe de la précision en fonction du rappel $p(r)$, dite de rappel/précision. La précision moyenne est la moyenne de la valeur de $p(r)$ sur l'intervalle $r = 0$ à $r = 1$:

$$\text{Précision moyenne} = \int_0^1 p(r)dr$$

En pratique, l'intégrale est remplacée par une somme finie :

$$\text{Précision moyenne} = \int_0^1 p(r)dr = \sum_{i=1}^{n} p(i)\Delta r(i)$$

où $\Delta r(i)$ est le changement de rappel entre $i-1$ et i.

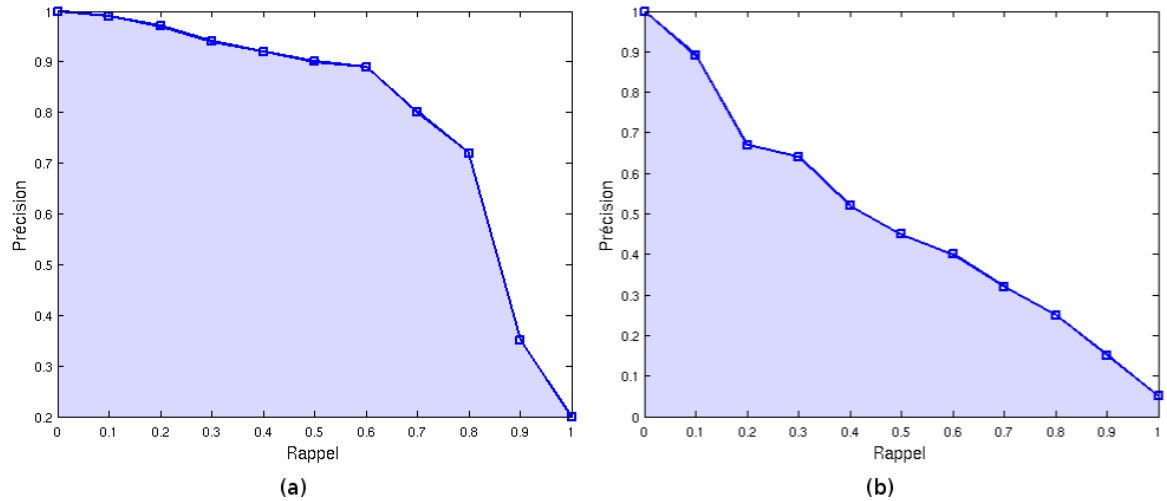


FIGURE 22 – Exemples de courbes rappel/précision, (a) précision moyenne de 78.9%, (b) précision moyenne de 48.5%

B.3 Nombre de faux positifs pour un pourcentage de positifs fixé

La précision moyenne est un moyen pour mesurer les performances d'un détecteur efficace, mais l'interprétation peut parfois être difficile. En effet, à l'utilisation, le but de notre algorithme est de détecter des symboles dans les cartes de Cassini, et comme tout détecteur le résultat sera imparfait et devra être filtré manuellement. La question qui se pose est donc "Combien d'exemples l'utilisateur devra filtrer manuellement si le détecteur à une précision moyenne de 70% ?". Pour ce faire nous introduisons une nouvelle mesure qui indique le nombre d'exemples négatifs retournés pour un certain pourcentage d'exemples positifs obtenus.

C

Développement JAVA

Cette annexe reprend en détail l'intégration du code JAVA utilisant les JKernelMachines [9], et permettant la sélection de négatifs difficiles durant l'apprentissage puis la détection d'objets dans les images.

C.1 Diagramme de classes

Le code a été intégré autour du patron ce conception logiciel *modèle-vue-contrôleur*. Le but étant de séparer le code en composants ayant chacun des interactions bien définies (voir figure 23) entre eux :

- Le **contrôleur** peut envoyer des commandes à sa vue associée pour mettre à jour la représentation visuelle du modèle. Il peut également envoyer des commandes au modèle pour mettre à jour ses états.
- Le **modèle** notifie ses changements d'état à ses vues et contrôleurs associés. Ces notifications permettent à la vue de produire une représentation visuelle mise à jour et au contrôleur de modifier les ensembles de commandes disponibles.
- La **vue** observe les informations du modèle pour générer une représentation visuelle à l'utilisateur.

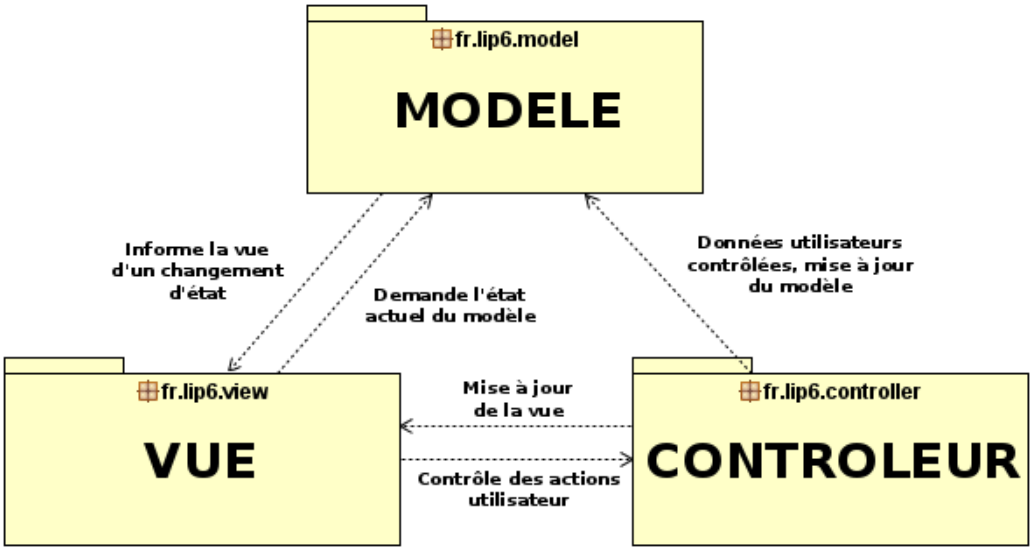


FIGURE 23 – Schéma du patron de conception modèle - vue - contrôleur

Les diagrammes de classes représentant l'architecture globale du code sont visibles figure 24 et 25

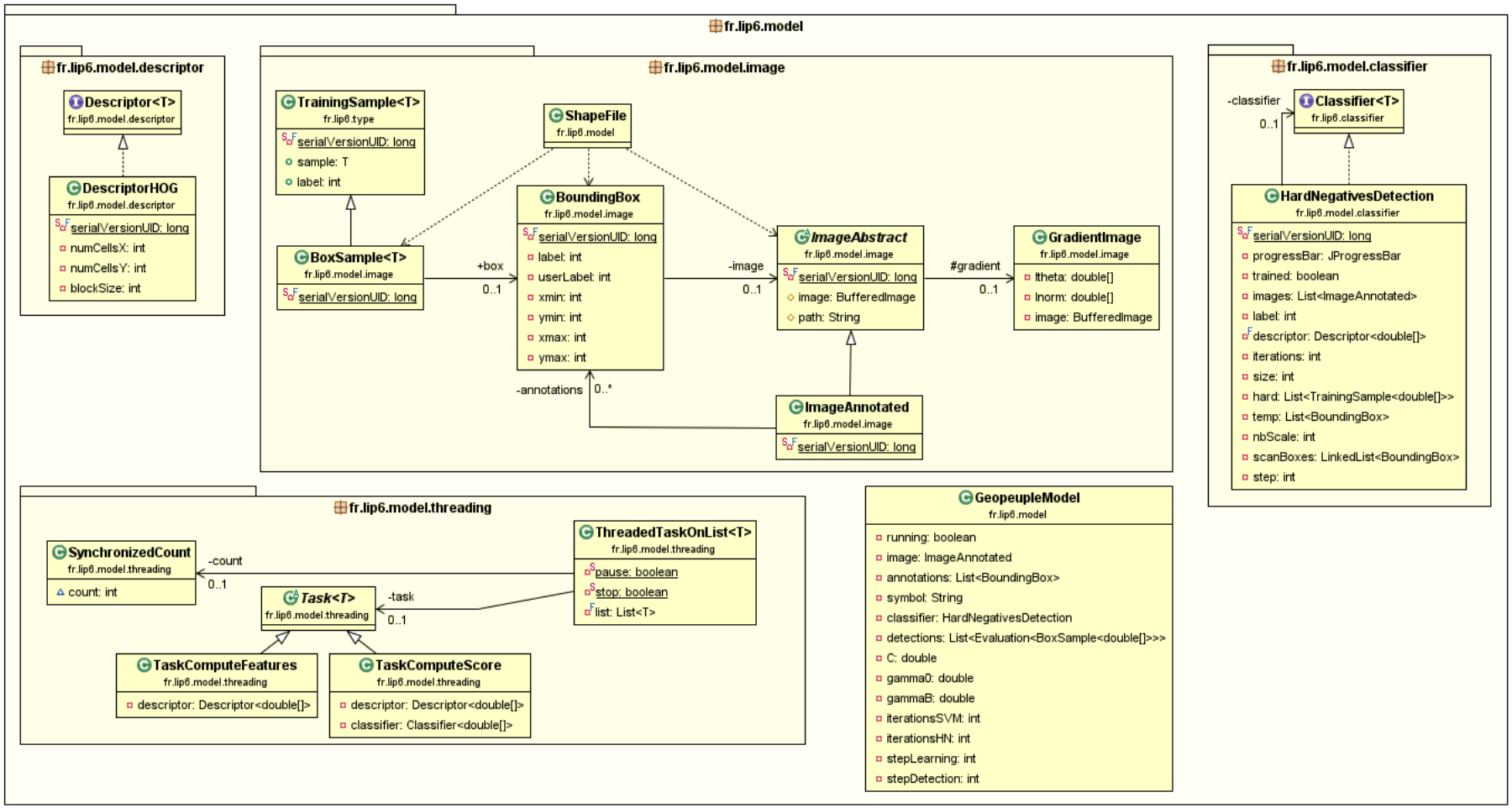


FIGURE 24 – Diagramme de classe de la partie modèle du code

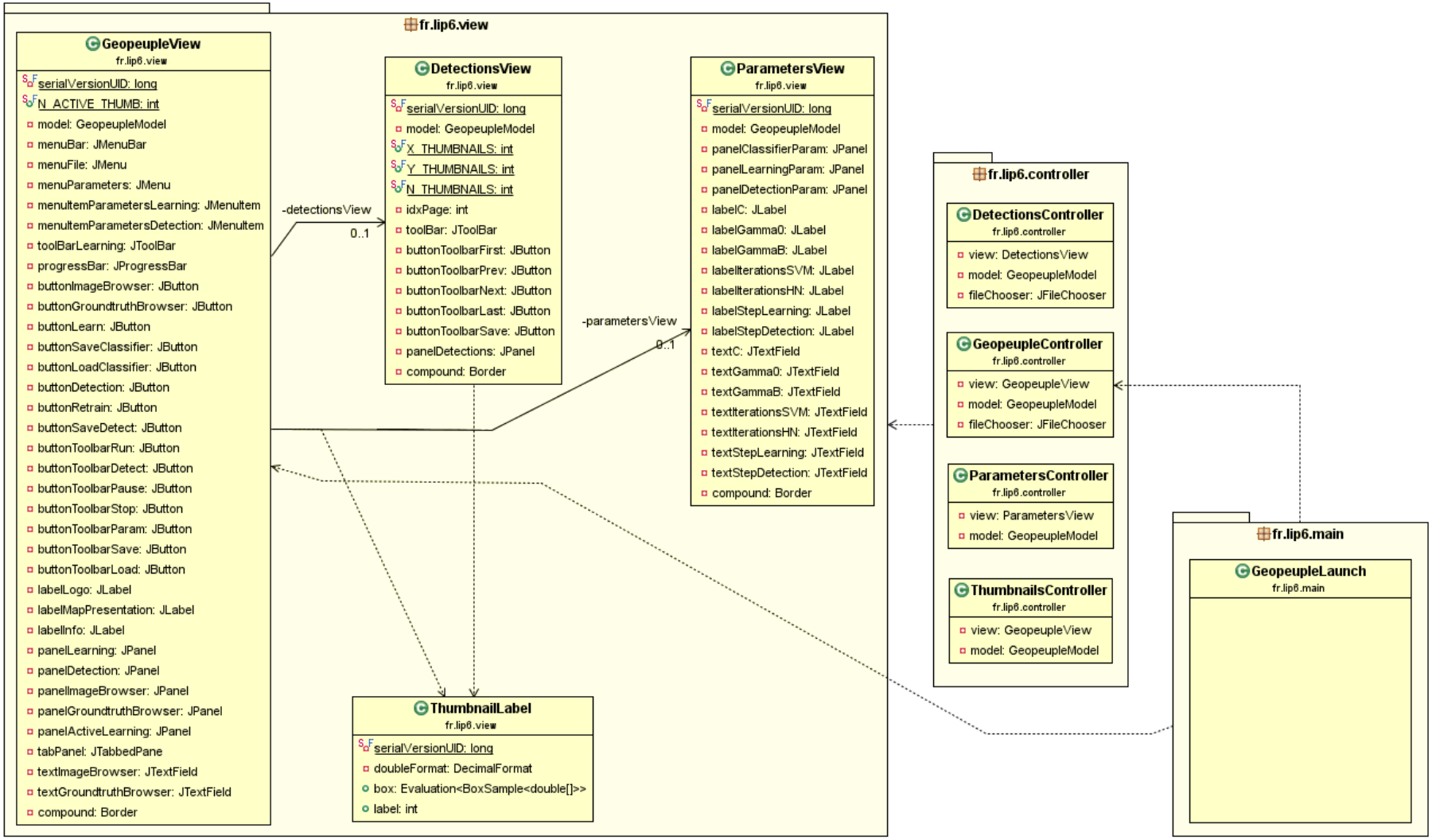


FIGURE 25 – Diagramme de classe de la partie vue et contrôleur du code

C.2 Description du code

Ce chapitre décrit plus en détail le fonctionnement interne du code en commençant par la partie modèle qui contient tout le coeur de l'algorithme.

C.2.1 Modèle

Cette section détaille les classes définissant le modèle du logiciel.

C.2.1.1 Package `fr.lip6.model.descriptor`

Ce package contient la déclaration d'un descripteur d'image général ainsi que ses implémentations.

`Descriptor.java`

```
public interface Descriptor<T> {

        public T compute(BoundingBox bb);
        public String getDescriptorType();

}
```

L'interface `Descriptor` définit un descripteur d'image. Une classe implémentant cette interface doit définir les méthodes `compute(BoundingBox bb)` et `getDescriptorType()`, qui retournent respectivement le descripteur de l'image contenu dans la `BoundingBox` donnée en paramètre, et le nom du descripteur.

La classe `DescriptorHOG` est l'implémentation de l'interface `Descriptor` pour le calcul des histogrammes de gradients orientés sur une image.

`DescriptorHOG.java`

```
public class DescriptorHOG
        implements Descriptor<double[]> {

        private int numCellsX;
        private int numCellsY;
        private int blockSize;

        public double[] compute(BoundingBox bb) {
                // Calcul des histogrammes de gradients orientés
        }

        public String getDescriptorType() {
                return "HOG";
        }
}
```

La méthode `compute` retourne un tableau de nombres réels représentant l'histogramme de gradients orientés calculé sur la `BoundingBox`.

C.2.1.2 Package `fr.lip6.model.image`

Ce package contient toutes les classes ayant trait à la gestion des images.

`ImageAbstract.java`

```
public abstract class ImageAbstract {

        protected transient BufferedImage image = null;
        protected transient GradientImage gradient = null;
        protected String path;

        public void load() {
                // Charge une image en mémoire
                ...
```

```
        }

        public void computeGradient() {
                // Calcule et stocke les gradients de l'image
                ...
        }

        public void unload() {
                // Déréférence les données
                ...
        }

        public List<BoundingBox> scan(BoundingBox box, int step) {
                // Récupère une liste de boîtes recouvrant l'image
                // selon un certain pas donné en paramètre
                ...
        }

        public List<Evaluation<BoxSample<double[]>>> detection(List<BoundingBox> boxes,
                        int step,
                        Classifier<double[]> classifier,
                        Descriptor<double[]> descriptor,
                        JProgressBar progressBar) throws InterruptedException {
                // Effectue une détection sur l'image
                ...
        }

}
```

La classe principale du package est la classe `ImageAbstract` qui stocke les données des images et permet d'effectuer des détections sur celles-ci.

- La méthode `computeGradient()` est appelée lors du chargement de l'image dans la méthode `load()` et soulève une exception `OutOfMemoryError` en cas de limitation de mémoire sur la machine. Cela n'empêchera pas le logiciel de fonctionner, mais celui-ci sera ralenti dans ses traitements.
- La méthode `scan()` permet de récupérer une liste de position d'une fenêtre glissante sous la forme d'une liste de `BoundingBox`. Cette méthode est utilisée en particulier lors de la construction d'une base de données pour l'apprentissage ou lors de l'appel à la méthode `detection()`.
- La méthode `detection()` effectue une détection de type fenêtre glissante sur une image. Les paramètres sont :
  - `List<BoundingBox> boxes` : la liste des fenêtres glissantes à utiliser pour effectuer la détection.
  - `int step` : le pas de détection en pixels. Pour une détection exhaustive `step=1`.
  - `Classifier<double[]> classifier` : le classifieur appris préalablement.
  - `Descriptor<double[]> descriptor` : le descripteur utilisé.
  - `JProgressBar progressBar` : une référence vers une barre de progression (facultatif).

La classe `ImageAnnotated` hérite de la classe `ImageAbstract` et définit une image possédant des annotations.

`ImageAnnotated.java`

```
public class ImageAnnotated
        extends ImageAbstract {

        private List<BoundingBox> annotations;

        public List<BoundingBox> extractNegatives(BoundingBox box, int step, int label) {
                // Extrait des négatifs d'une image
                ...
        }

}
```

Les annotations sont stockées par une liste de `BoundingBox` ayant leur `label` à `1`. La méthode aditionnelle

`extractNegatives()` permet d'extraire une liste de fenêtre ne se superposant pas avec les annotations. La classe `GradientImage` permet de calculer et stocker les gradients d'une image.

GradientImage.java

```
public class GradientImage {

        private double[] Itheta;
        private double[] Inorm;
        private BufferedImage image;

        public void compute(BufferedImage image) {
                // Calcule les gradients d'une image
                ...
        }

        public double[] getTheta(BoundingBox box) {
                // Récupère l'angle des gradients de chaque pixel
                // contenu dans une BoundingBox
                ...
        }

        public double[] getNorm(BoundingBox box) {
                // Récupère la norme des gradients de chaque pixel
                // contenu dans une BoundingBox
                ...
        }
}
```

La classe `BoundingBox` définit une sous-partie d'une image.

BoundingBox.java

```
public class BoundingBox {

        private ImageAbstract image;

        private int label;
        private int userLabel = 0;

        private int xmin;
        private int ymin;
        private int xmax;
        private int ymax;

        public double overlap(BoundingBox box) {
                // Retourne le pourcentage de superposition
                // entre cette boîte et box
                ...
        }

        public double contains(BoundingBox box) {
                // Retourne le pourcentage de la boîte boxes
                // contenu dans cette boîte
                ...
        }

}
```

Une `BoundingBox` est définie par ses coordonnées dans l'image `xmin`, `ymin`, `xmax` et `ymax`. La référence de l'image à laquelle appartient la `BoundingBox` est stockée dans la variable `image`. Les méthodes `overlap()` et `contains()` permettent de calculer différentes valeurs de recouvrement entre deux `BoundingBox` (voir figure 26) :

- `overlap(BoundingBox box)` : est égal à la zone de recouvrement entre les deux `BoundingBox` divisé par la surface totale des deux `BoundingBox`. C'est le calcul général du recouvrement entre deux boîtes. La valeur retournée est comprise entre 0 (aucun recouvrement) et 1 (recouvrement parfait entre deux boîtes de la même dimension).
- `contains(BoundingBox box)` : est égal à la zone de recouvrement entre les deux `BoundingBox` divisé par la surface de `box`. Cela corresponds à "est-ce que `box` est contenue dans la boîte courante ?".

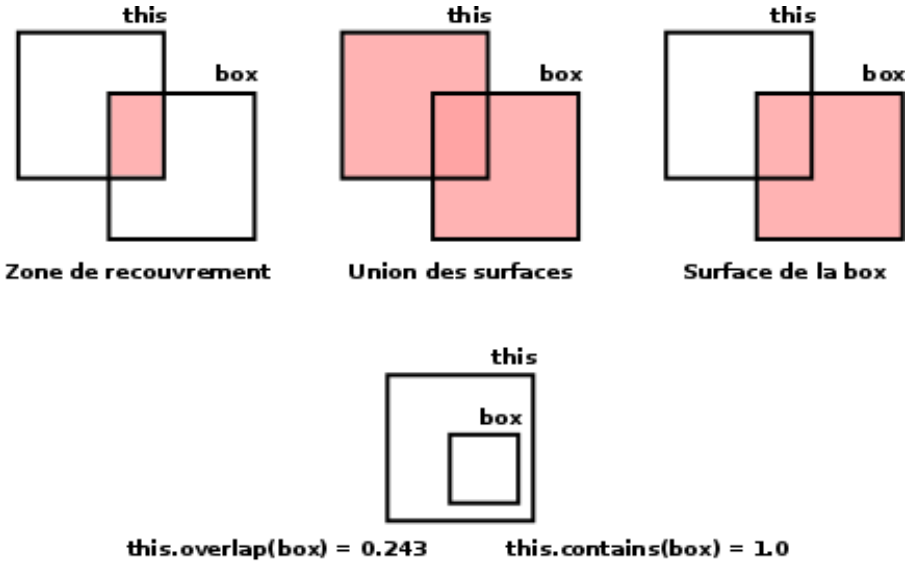


FIGURE 26 – Recouvrement entre deux `BoundingBox`

L'interface `Fusion` définit un algorithme de fusion de `BoundingBox`. Une classe implémentant cette interface doit définir la méthode `fusion()` qui prends en paramètre une liste de `BoundingBox` et supprime les moins importantes.

Fusion.java

```
public interface Fusion {

        public void fusion(List<Evaluation<BoxSample<double[]>>> boxes);

}
```

Les classes `FusionNMS`, `FusionMean`, `FusionMax` et `FusionContain` implémentent cette interface et suppriment les `BoundingBox` comme indiqué figure 27

C.2.1.3 Package `fr.lip6.model.threading`

Ce package contient toutes les classes ayant trait à la gestion de la parallélisation des calculs.
La classe `SynchronizedCount` est un simple compteur qui permettra d'effectuer des tâches sur des listes de manière à ne créer aucun conflit à l'aide du mot clef `synchronized`.

SynchronizedCount.java

```
public class SynchronizedCount {

        private int count;

        public void reset() { count = 0; }
        public void inc() { count++; }
        public void dec() { count--; }

}
```

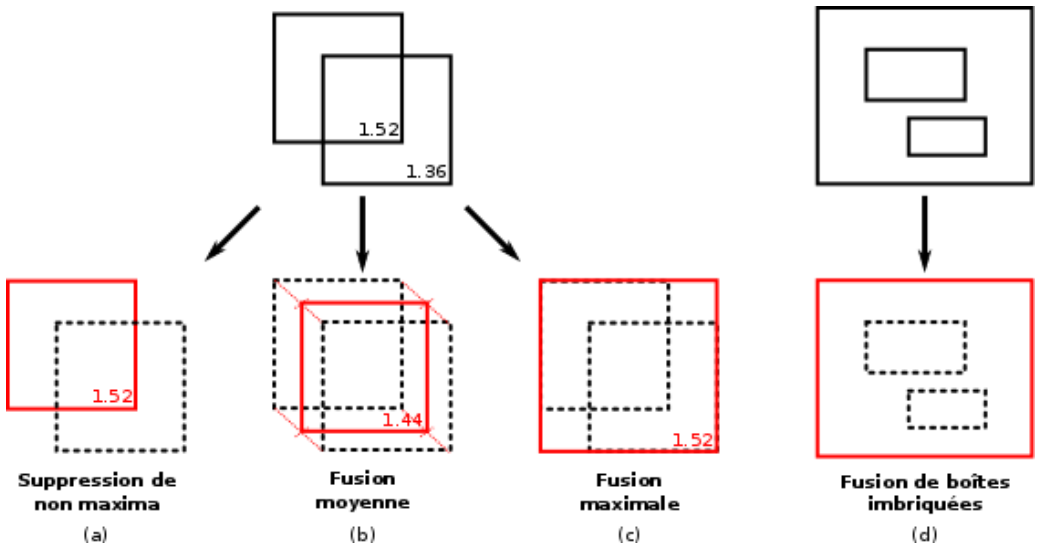


FIGURE 27 – Fusion de `BoundingBox`, (a) `FusionNMS`, (b) `FusionMean`, (c) `FusionMax`, (d) `FusionContain`

La classe `Task` défini une tâche à effectuer sur une liste d'objet de type `T`.

`Task.java`

```
public abstract class Task<T> {

        public abstract void perform(T t);
        public abstract String type();

}
```

Une classe descendant de la classe `Task` devra implémenter les méthodes `perform()` et `type()` :

- `void perform(T t)` : la méthode intégrant la tâche à effectuer sur un objet de type `T`.
- `String type()` : retourne le nom de la tâche effectuée.

Les classes `TaskComputeFeatures` et `TaskComputeScore` sont toutes deux des implémentations de la classe `Task` et permettent respectivement de calculer les descripteurs et les scores d'une liste de `BoundingBox`.

La classe `ThreadedTaskOnList` prends une `Task` et une liste d'objet `T` en paramètre, et effectue la tâche sur la liste de manière parallèle.

`ThreadedTaskOnList.java`

```
public class ThreadedTaskOnList<T> {

        private static boolean pause = false;
        private static boolean stop = false;

        private final List<T> list;
        private final Task<T> task;
        private final SynchronizedCount count;

        public void start() throws InterruptedException {
                // Lance l'ensemble des threads de calcul
                ...
        }

        public static void pause() {
                // Met en pause la tâche en cours
```

```
        }

        public static void unpause() {
                // Reprends la tâche
        }

        public static void stop() {
                // Arrête le calcul
        }

}
```

C.2.1.4 Package `fr.lip6.model.classifier`

Ce package contient toutes les classes utilisées pour la classification des fenêtres.
La classe `HardNegativesDetection` permet l'apprentissage d'un classifieur en sélectionnant à chaque itération les exemples les plus informatifs.

`HardNegativesDetection.java`

```
public class HardNegativesDetection
        implements Classifier<double[]> {

        private Classifier<double[]> classifier;
        private Descriptor<double[]> descriptor;

        private int label;
        private transient List<ImageAnnotated> images;

        private int iterations = 5;
        private int size = 5000;
        private int step = 10;
        private int nbScale = 3;
        private List<BoundingBox> scanBoxes;

        private List<TrainingSample<double[]>> hard;

        public void train(List<TrainingSample<double[]>> l) {
                // L'apprentissage s'effectue ici
                ...
        }

}
```

Les attributs de ce classifieur sont les suivants :

- `Classifier<double[]> classifier` : classifieur utilisé pour l'apprentissage.
- `Descriptor<double[]> descriptor` : descripteur utilisé pour le calcul des features de chaque fenêtre.
- `int label` : label de la classe à apprendre, dans le cas où les images annotées contiennent plusieurs labels différents.
- `List<ImageAnnotated> images` : liste des images annotées.
- `int iterations` : nombre d'itérations de sélection d'exemples informatifs.
- `int size` : taille maximum d'exemples retenus.
- `int step` : pas utilisé pour scanner les images annotées.
- `int nbScale` : nombre d'échelle utilisée, dans le cas où on effectue une détection multi-échelle.
- `List<BoundingBox> scanBoxes` : fenêtres glissantes utilisées, dans le cas où on effectue une détection multi-ratio.
- `List<TrainingSample<double[]» hard` : liste des exemples les plus informatifs retenus.

C.2.2 Vue

Le package `fr.lip6.view` contient toutes les classes permettant la gestion de l'interface graphique.

- `GeopeupleView.java` : gestion de l'affichage de la fenêtre principale de l'interface graphique.
- `ParametersView.java` : gestion de l'affichage de la fenêtre de paramètres.
- `DetectionsView.java` : gestion de l'affichage de la fenêtre de sélection manuelle des exemples.
- `ThumbnailLabel.java` : gestion de l'affichage d'une icône annotable manuellement.

C.2.3 Contrôleur

Le package `fr.lip6.controller` contient toutes les classes permettant la gestion des commandes transmises depuis la vue jusqu'au modèle.

- `GeopeupleController.java` : gestion des commandes transmises depuis la fenêtre principale.
- `ParametersController.java` : gestion des modifications des paramètres du modèle.
- `DetectionsController.java` : gestion de la mise à jour du label utilisateur dans la fenêtre de sélection manuelle des exemples.
- `ThumbnailController.java` : gestion des annotations.

D

Livrable - Tutoriel d'utilisation

D.1 Présentation générale

FIGURE 28 – Présentation de l'onglet apprentissage

1. Onglets de sélection entre le mode **apprentissage** et le mode **détection**.
2. Barre d'outils (décrite plus en détail figure 30).
3. Sélection et chargement d'une carte de Cassini.
4. Sélection et chargement d'un fichier d'annotations.
5. Bouton d'exécution de l'apprentissage.
6. Bouton de sauvegarde du détecteur préalablement appris.
7. Fenêtre d'information.

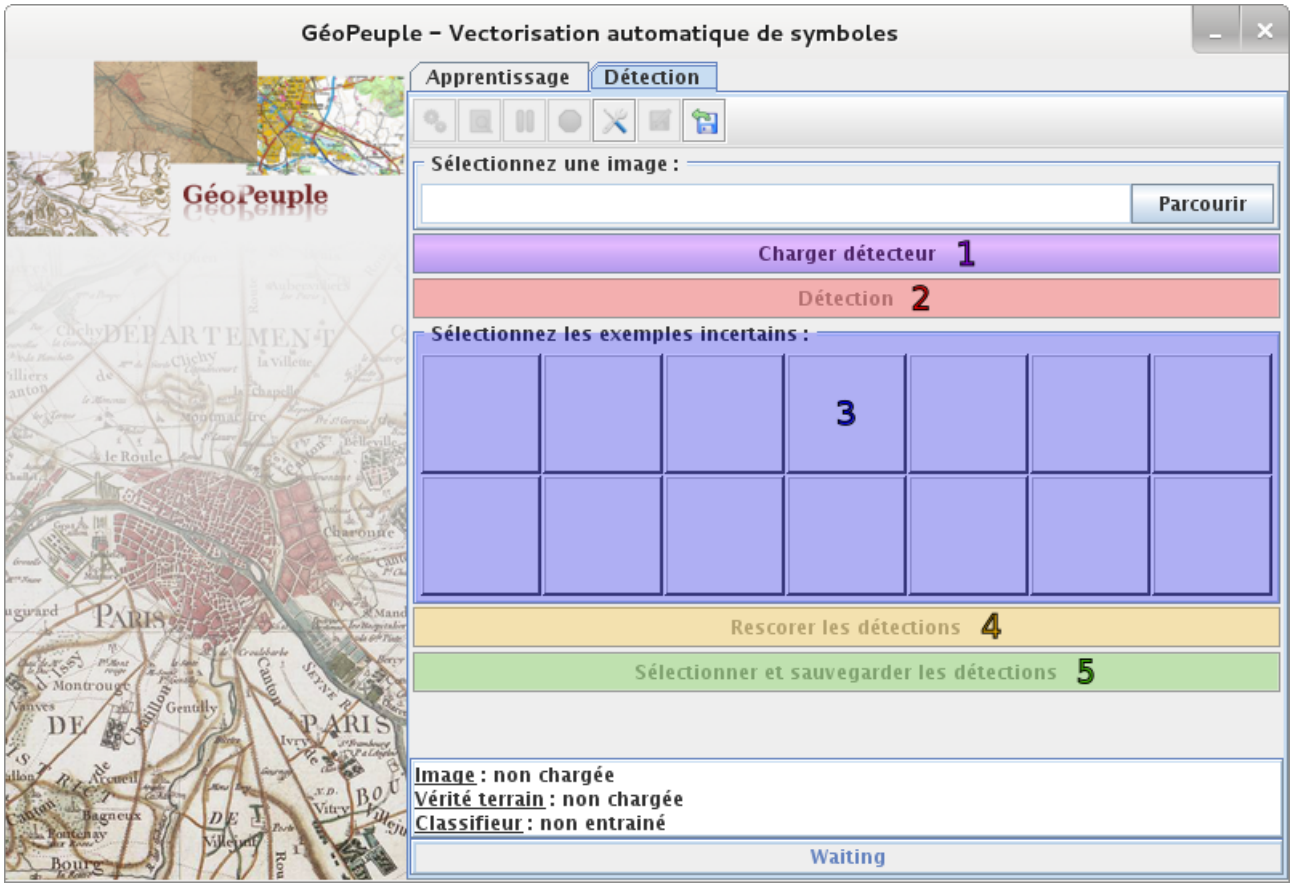


FIGURE 29 – Présentation de l'onglet détection

1. Permet de charger un détecteur préalablement entraîné.
2. Effectue une détection sur la carte chargée.
3. Fenêtre de sélection des exemples incertains après un premier passage du détecteur sur la carte. Cela permet de raffiner la décision finale du détecteur manuellement.
4. Met à jour le score des détections en fonction des annotations manuelles.
5. Bouton de sélection et de sauvegarde de la liste des détections au format shapefile.

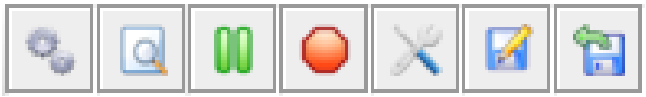


FIGURE 30 – Présentation de la barre d'outils

1.  Effectue l'apprentissage.
2.  Effectue une détection.
3.  Met en pause un traitement.
4.  Arrête le traitement en cours.
5.  Ouvre le panneau des paramètres (figure 31).
6.  Sauvegarde un détecteur
7.  Charge un détecteur.

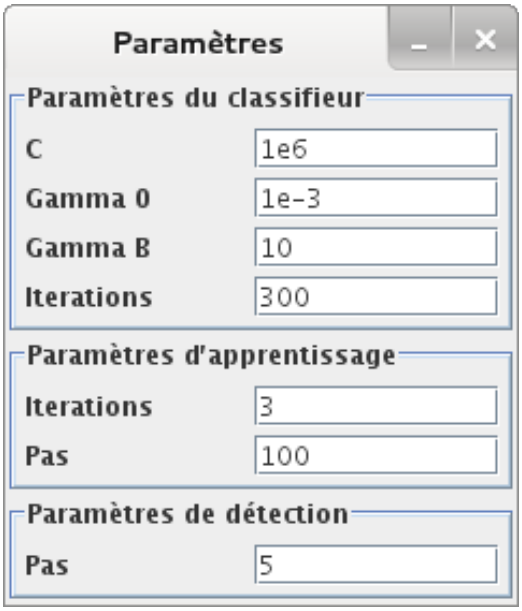


FIGURE 31 – Présentation du panneau de paramétrage

D.2 Entraîner un détecteur

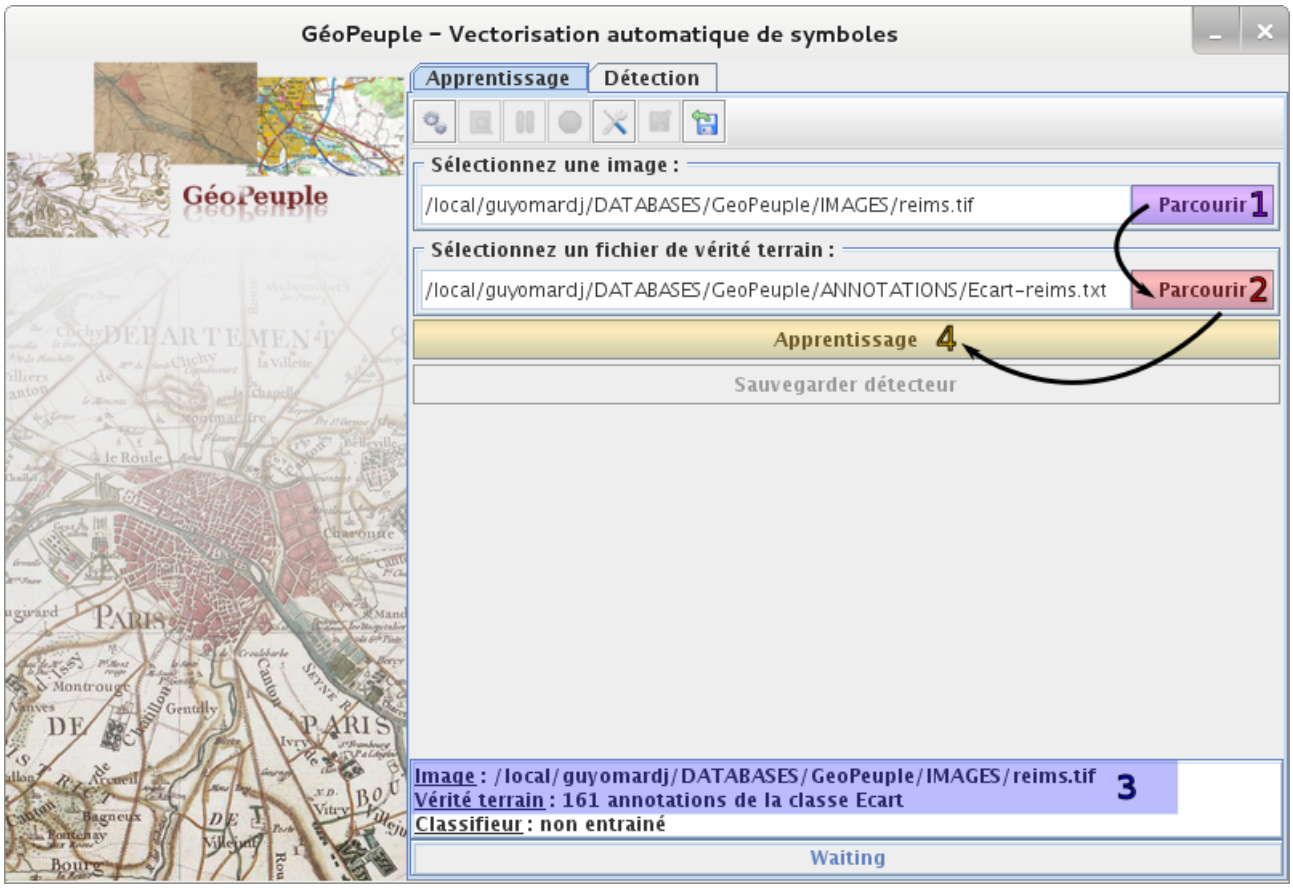


FIGURE 32 – Présentation de l'apprentissage d'un classifieur

1. Sélectionnez une image à l'aide du bouton **Parcourir**. En cas de problème lors du chargement de l'image, un message d'erreur apparaîtra.
2. Sélectionnez un fichier d'annotations à l'aide du bouton **Parcourir**. En cas de problème lors du chargement du fichier, un message d'erreur apparaîtra.
3. Si le chargement de l'image et du fichier d'annotations s'est correctement déroulé, les informations se mettent à jour automatiquement.
4. Si le chargement de l'image et du fichier d'annotations s'est correctement déroulé, le bouton **Apprentissage** passera en sur-brillance. Cliquez dessus pour lancer l'apprentissage du détecteur (Cette étape peut être relativement longue en fonction de la puissance de calcul de la machine).

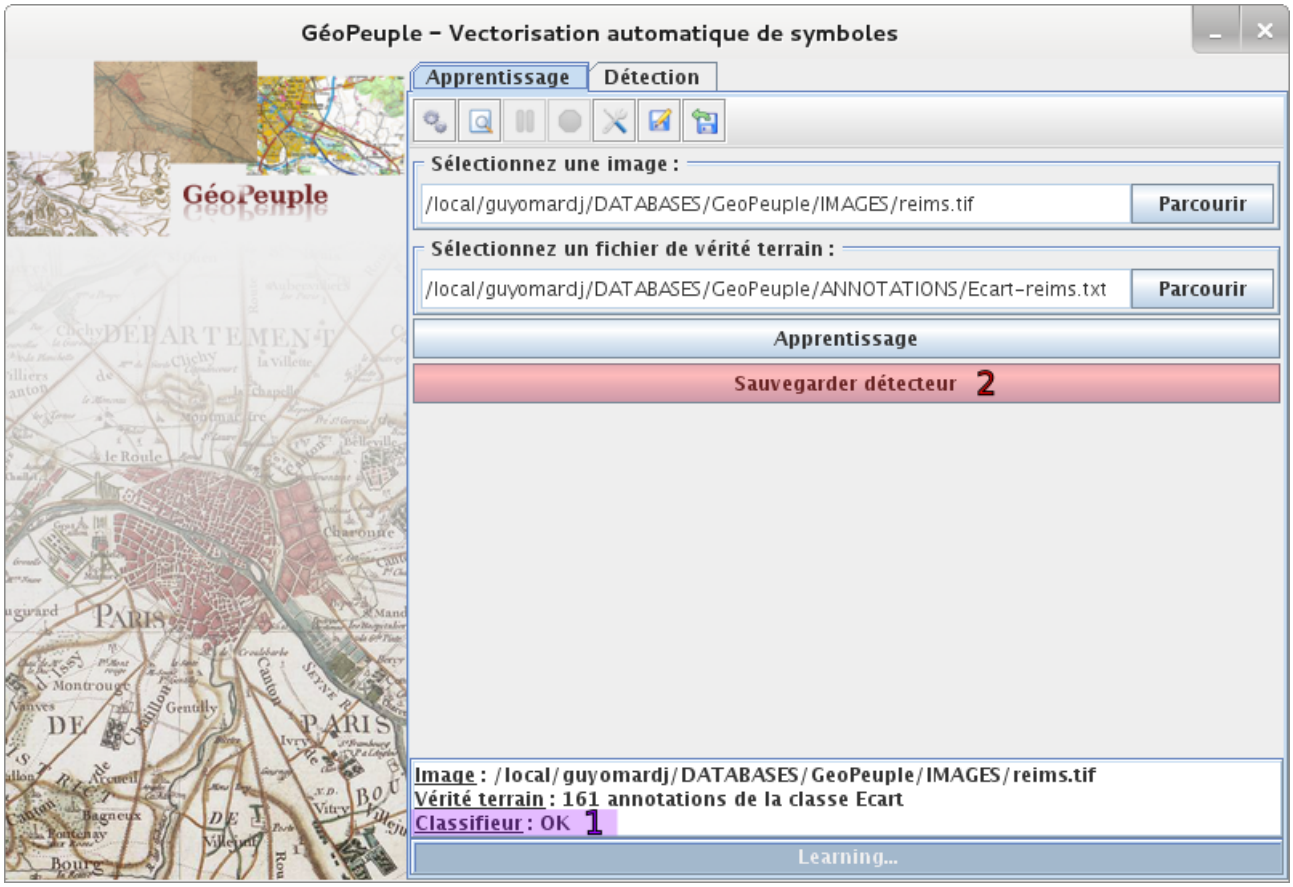


FIGURE 33 – Sauvegarde d'un classifieur après apprentissage

1. Si l'apprentissage s'est correctement déroulé, le classifieur s'affiche comme correctement appris dans la fenêtre d'information.
2. Si l'apprentissage s'est correctement déroulé, le bouton **Sauvegarder détecteur** passera en sur-brillance, et permettra d'enregistre le classifieur pour une utilisation ultérieure.

D.3 Effectuer une détection

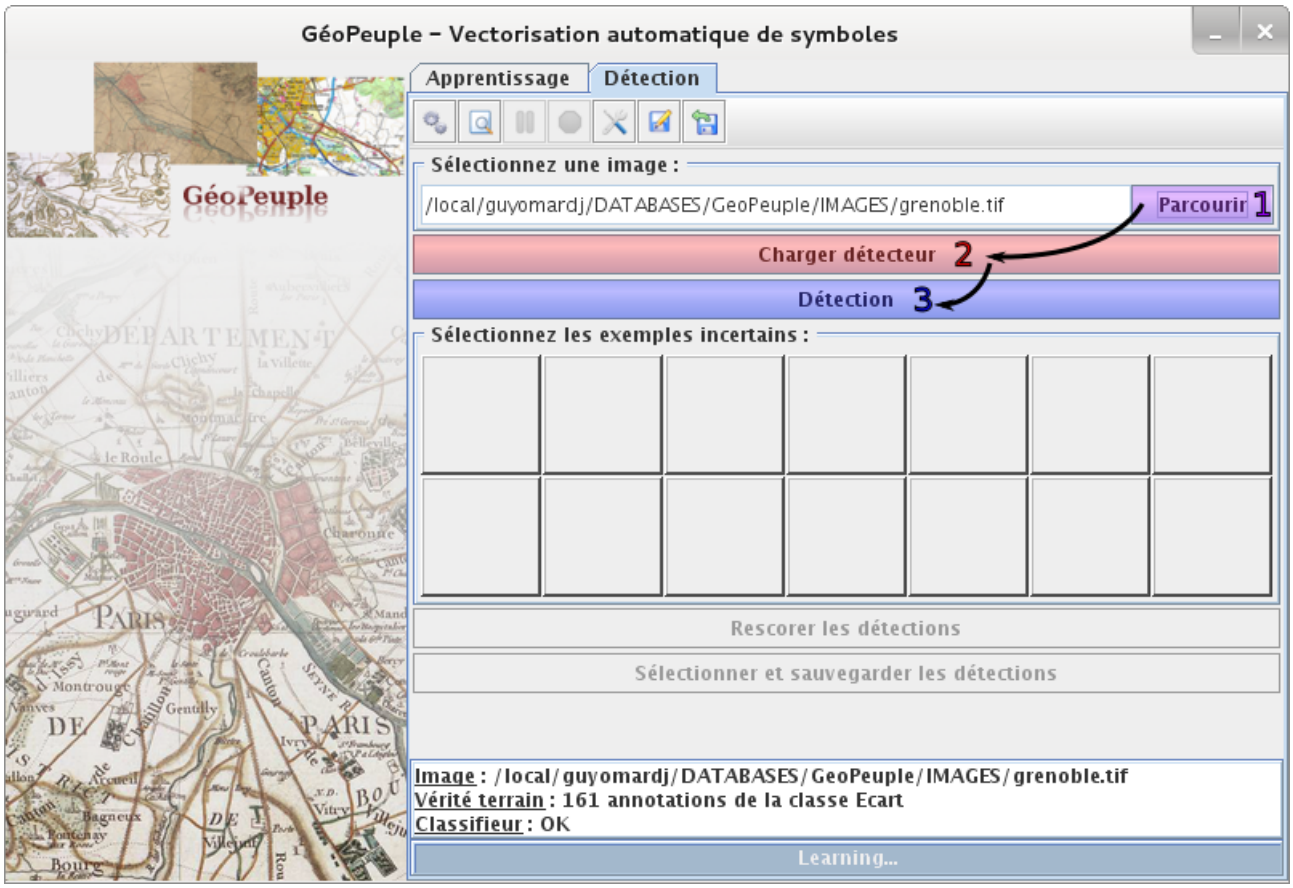


FIGURE 34 – Chargement d'un détecteur en vue d'une détection

1. Sélectionnez une nouvelle image à l'aide du bouton **Parcourir**.
2. Chargez un détecteur préalablement appris ou fournis.
3. Cliquez sur le bouton détection afin d'effectuer une première étape de vectorisation (Cette étape peut être relativement longue en fonction de la puissance de calcul de la machine).

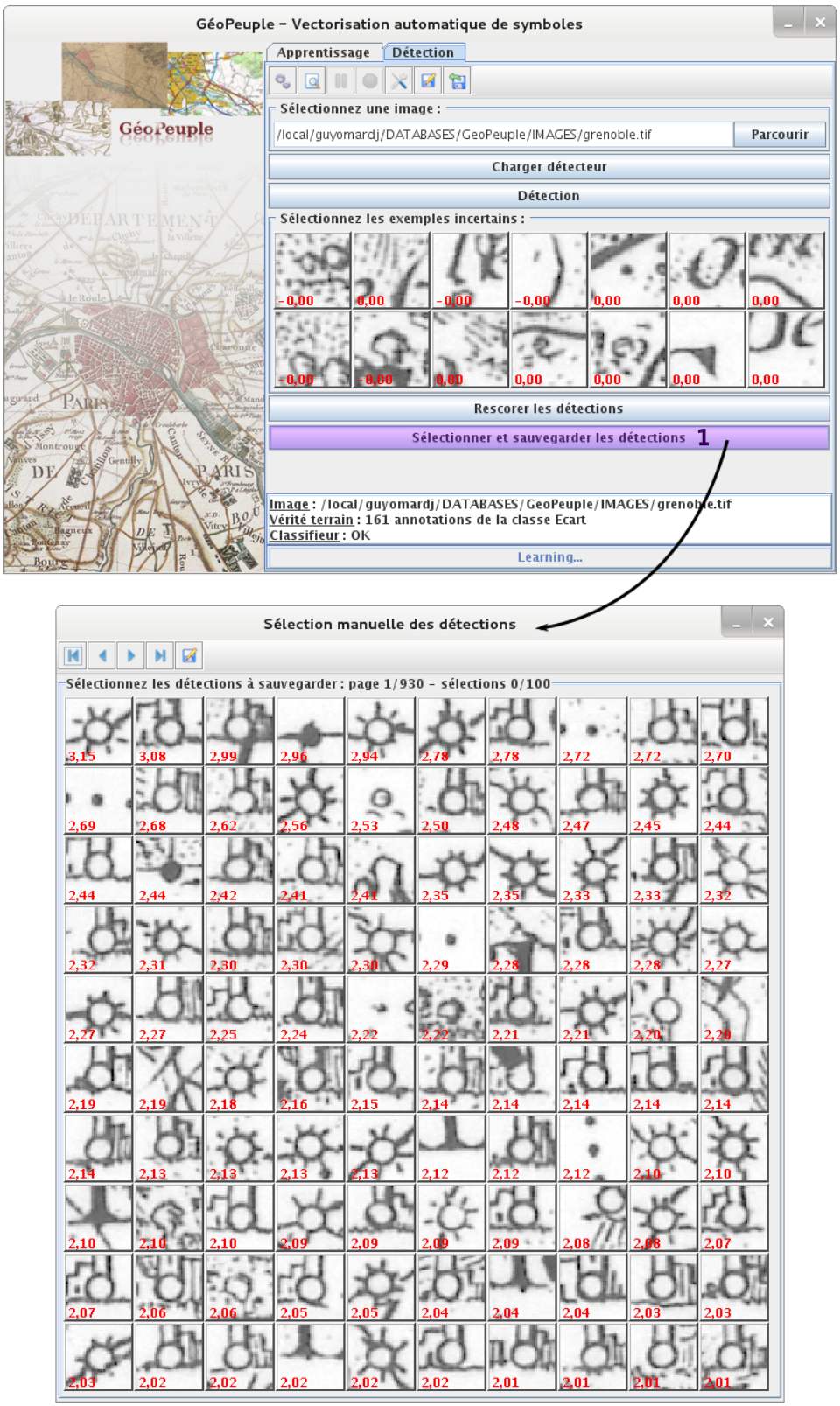


FIGURE 35 – Ouverture du panneau de sélection des détections finales

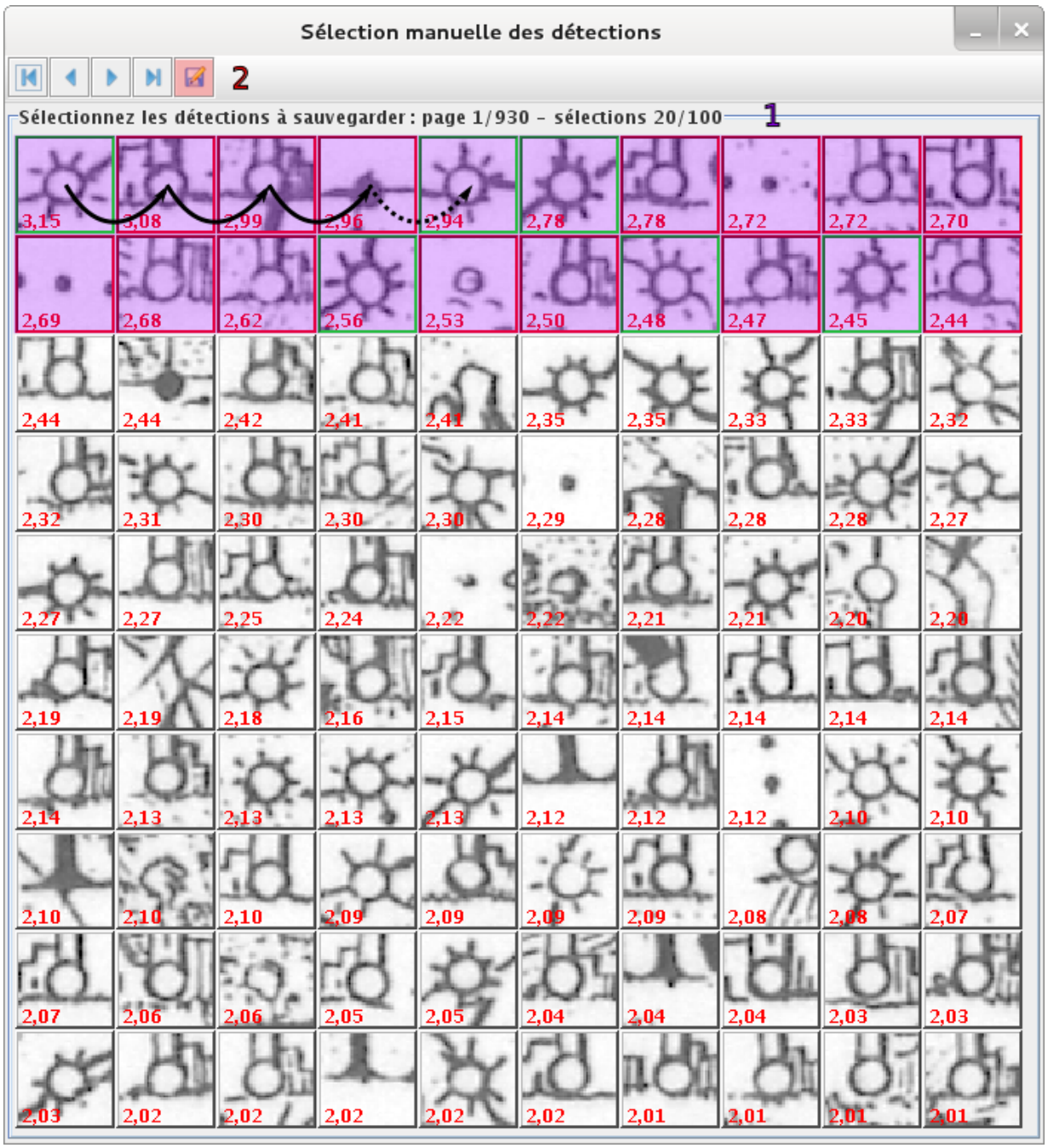


FIGURE 36 – Sélection et sauvegarde des détections finales

1. Sélectionnez les exemples à conserver dans le fichier final. Un **clic gauche** permet d'annoter l'exemple comme appartenant à la classe recherchée, un **clic droit** permet de l'annoter comme étant un faux positif.
2. Sauvegardez les détections au format shapefile.

D.4 Améliorer le classement des détections

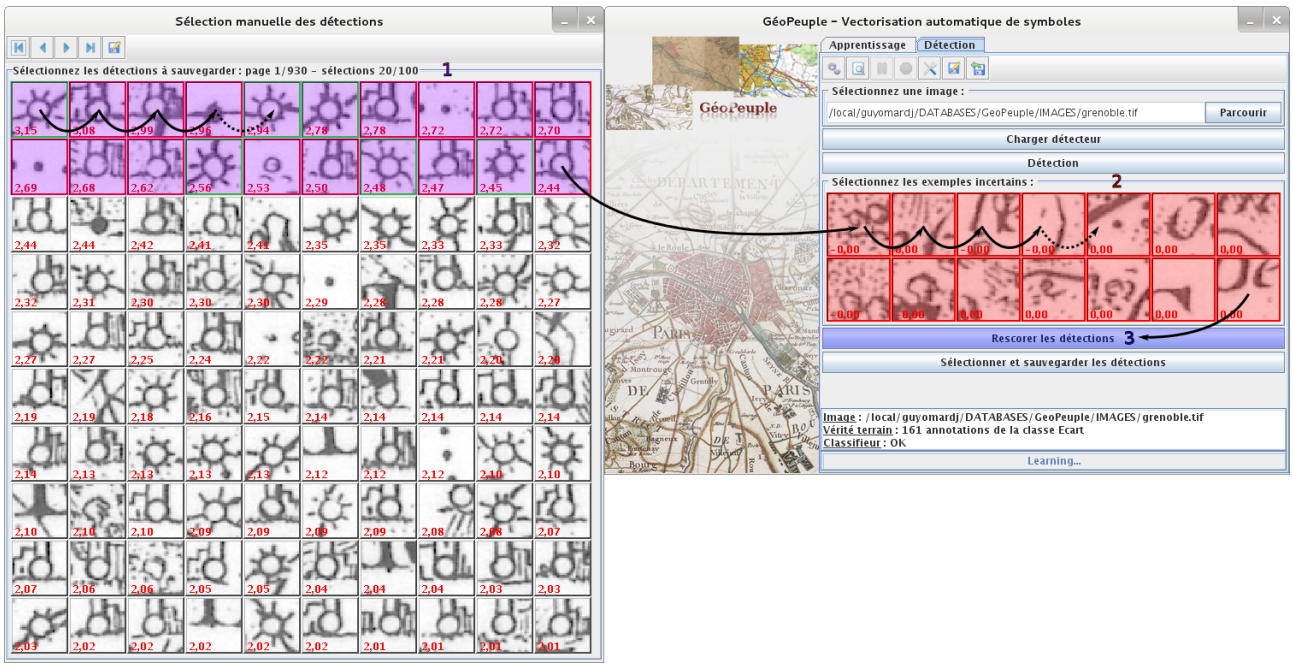


FIGURE 37 – Indication manuelle des exemples positifs et négatifs à ajouter au détecteur

1. Vous pouvez sélectionner manuellement des exemples à ajouter au détecteur dans la fenêtre de sauvegarde des détections. **Attention**, les exemples annotés négativement sont définitivement supprimés de la liste après avoir recalculer les scores.
2. Vous pouvez sélectionner manuellement des exemples à ajouter au détecteur parmi ceux proposé par l'algorithme de détection.
3. Cliquez sur le bouton **Rescorer les détections** afin d'améliorer la qualité de la détection.

Cette étape peut être effectuer autant de fois que nécessaire afin de faciliter autant que possible le travail de l'utilisateur pour annoter l'ensemble des détections.

Références

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

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

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http ://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[5] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9) :1627–1645, 2010.

[6] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, 1997.

[7] Geremy Heitz and Daphne Koller. Learning spatial context : Using stuff to find things. In *Proceedings of the 10th European Conference on Computer Vision : Part I*, ECCV '08, pages 30–43, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Yang Mingqiang, Kpalma K. Idiyo, and Ronsin Joseph. A survey of shape feature extraction techniques. *Pattern Recognition, Peng-Yeng Yin (Ed.) (2008) 43-90*, pages 43–90, November 2008.

[9] David Picard, Nicolas Thome, and Matthieu Cord. JKernelMachines, 2013. http ://mloss.org/software/view/409/.

[10] Andrew Rabinovich, Andrea Vedaldi, Carolina Galleguillos, Eric Wiewiora, and Serge Belongie. Objects in context. In *ICCV*, Rio de Janeiro, 2007.

[11] James H. Hays Alexei A. Efros Martial Hebert Santosh K. Divvala, Derek Hoiem. An empirical study of context in object detection. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2009.

[12] Alain Trémeau, Shoji Tominaga, and Konstantinos N. Plataniotis. Color in image and video processing : most recent trends and future research directions. *J. Image Video Process.*, 2008 :7 :1–7 :26, January 2008.