Outils développés

5.1. Motivations

Au centre de robotique, nous utilisons pour nos recherches en aide avancée à la conduite automobile (ADAS), le logiciel RTMaps qui nous permet d'enregistrer et de rejouer de façon synchroniser des séquences d'acquisition des données des capteurs de nos véhicules (cf. section 3.6.1 de ce document). Ce logiciel, initialement développé au centre à partir d'un besoin d'un outils d'acquisition des données (data logging) et de traitement dans des conditions proches de l'embarqué, nous facilite grandement la tâche dans le fait de pouvoir prototyper nos algorithmes sur nos ordinateurs de bureau à partir des séquences acquises comme si il s'agissait de tests en temps réel lors d'essai en véhicule, puis d'une simple formalité, de les porter sur nos véhicules en copiant les fichiers générés par ce logiciel sur les ordinateurs embarqués.

Les différents modules mis en œuvre dans les systèmes ADAS, et notamment ceux du laboratoire, se basent généralement sur des algorithmes de détections (détection des lignes de marquage au sol, détection des véhicules, détection des panneaux signalétiques, détection des feux tricolores, détection des piétons,…). Ces détections se font par traitement du signal du capteur soit par des algorithmes très spécifiques au problème (ex : détection des cercles dans les images provenant d'une caméra) soit par l'utilisation d'algorithmes du domaine de l'intelligence artificielle dits d'apprentissage numérique (ex : les réseaux de neurone).

L'utilisation répétitive de ces algorithmes au cours de travaux de recherche, et l'exploitation de nombreuses séquences d'acquisitions avec le logiciel RTMaps, amène à se poser plusieurs questions :

- Comment évaluer efficacement nos divers algorithmes de détection mis en place ?
- Comment améliorer ces algorithmes d'apprentissage numérique par l'apport de nouvelles idées ?
- Comment organiser ces giga octets de données issues des différents scénarios d'acquisitions des séquences de tests en véhicule ?

Ces questions que je me suis posées durant mes travaux de thèse m'ont amené à développer et à proposer plusieurs outils présentés dans les trois prochain paragraphes.

Cette présentation pourrait plus ressembler, pour certains outils, aux yeux du lecteur à une description de type ingénierie (au contraire d'un travail de recherche), mais elle a bien sa place dans ce manuscrit de thèse car ces outils développés constituent un vrai travail de réflexion sur des problèmes de recherche, auxquels tout chercheur a dû être confronté à un moment donné. En effet, toutes les questions évoquées ici, trouvent leurs réponses dans chacun de ces logiciels qui

correspondent à un vrai besoin pour l'évaluation et l'analyse des paramètres des algorithmes développés au cours d'un travail de recherche.

5.2. Outils d'édition de vérité terrain

5.2.1. Présentation générale

Une autre question soulevée au cours de mes travaux de thèse était de savoir comment évaluer efficacement un algorithme de détection, et notamment celui des panneaux de limitation de vitesse mis en place dans les travaux présentés dans ce mémoire.

Il est certes possible d'appliquer l'algorithme développé sur une longue séquence (scénario) de test et de relever manuellement les résultats (i.e. le nombre de bonnes et de non détections) mais cela s'avère très fastidieux et peu précis. En effet, au cours du travail de recherche et de développement d'une méthode de détection (ou autre), les algorithmes mis en place évoluent forcément soit par le changement des jeux de paramètres utilisés afin d'affiner au mieux les résultats, soit tout simplement parce que de nouvelles méthodes sont implémentées. Idéalement, après chaque tentative d'amélioration d'un processus, il faudrait évaluer l'apport des changements afin d'être sûr que ces derniers améliorent bien les résultats. Or il est clair que pour évaluer systématiquement son algorithme après chaque modification sur un scénario de test, même court d'une dizaine de minutes, il est nécessaire d'automatiser ces tests.

Il existe de nombreux logiciels d'annotation vidéo disponibles. Le plus connu et le plus utilisé d'entre eux est sans doute le logiciel Viper - Video Performance Evaluation Resource (Doermann, et al., 2000). Mais on peut aussi citer les logiciels Anvil - Video Annotation Research Tool (Kipp, 2001), ODViS - Open Development Environment for Evaluation of Video Surveillance Systems (Jaynes, et al., 2002),… Une liste non exhaustive mais assez complète peut être trouvée dans (Travis Rose, 2007).

Dans ces travaux, (Travis Rose, 2007) relève de nombreux points sur l'impossibilité d'utiliser un outil existant pour l'annotation de ces séquences de travail, dont entre autres :

- Les outils existants ne fournissent pas toutes les fonctionnalités voulues.
- Il n'y a pas de support pour des formats vidéos non standards.
- Ils ne supportent pas l'annotation de vidéo de plus de 10 min.
- Ils ne supportent pas l'annotation de vidéo issues de plusieurs caméras
- Ils ne supportent pas des synchronisations précises entre de multiples vidéos

Bien que certains de ces points soient discutables, il est aisé de relever un certains nombre de ces points dans le cas des séquences ^{rt}Maps : le support des vidéos ^{rt}Maps ne sera jamais pris en compte par les autres logiciels d'annotation vidéo, ils ne supportent pas les synchronisations précises entre de multiples sources, et aussi, ils ne fourniront pas l'ensemble des fonctionnalités voulues.

De ce constat est né SamGT (GT étant l'acronyme de Ground Truth) : un logiciel capable de lire et d'annoter des séquences ^{rt}Maps afin de réaliser des fichiers dits de vérité terrain (*ground truth* en

anglais, fichiers contenant la position des objets à détecter pour chacune des images d'une vidéo), couplés à un package rtMaps (un ensemble de modules) pouvant comparer automatiquement les résultats d'un algorithme par rapport au fichier de vérité terrain préalablement édité par un opérateur humain. Tout l'intérêt et l'originalité résident dans l'utilisation du package rtMaps, qui, comme son nom l'indique, fonctionne nativement sous notre plateforme de prototypage, l'algorithme mis en place n'aura donc pas besoin d'être adapté (i.e. aucun besoin de traduire les résultats dans un format donné) afin de pouvoir être évalué avec SamGT.

5.2.2. L'édition de fichier de vérité terrain

a) L'éditeur

La conception de l'éditeur de fichier de vérité terrain du logiciel SamGT est, à l'instar de la conception d'un algorithme sous rtMaps, réalisé de façon modulaire. Ainsi comme le résume la Figure 174, cet éditeur est conçu de façon à n'être en fait qu'un « moteur » récupérant les images frame par frame dans un flux vidéo et appliquant une série de plugins (i.e. d'algorithmes) sur une image (qui sera donc modifiée) avant de l'afficher.

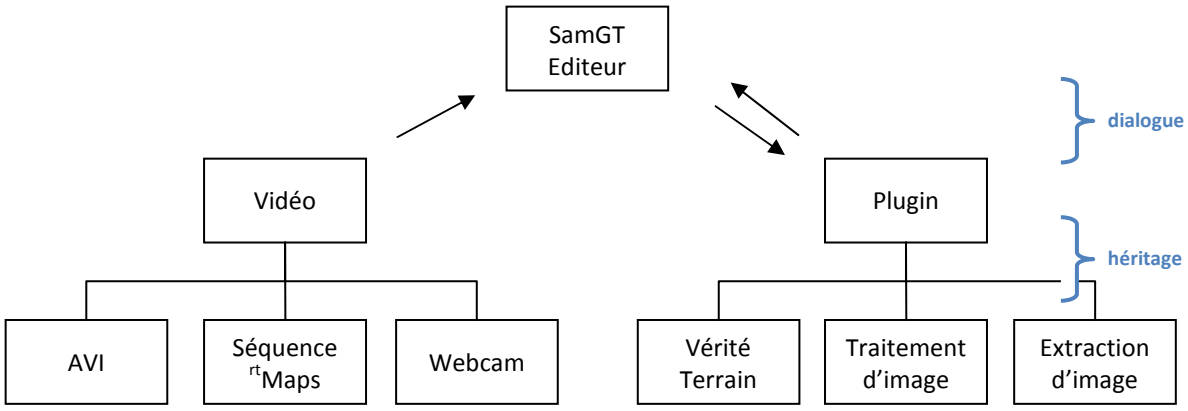


Figure 174 – Schéma de fonctionnement modulaire de l'éditeur SamGT

Les différents types de flux vidéo reconnus par l'éditeur (qui est, pour rappel, un logiciel indépendant de la plateforme rtMaps) sont bien entendu ceux supportés nativement par rtMaps (des formats vidéo propriétaires gérant la datation de chaque image à la microseconde) tel que les formats RAW et JSEQ, mais aussi ceux pouvant être habituellement utilisés sur ordinateur tel que le format AVI et les images vidéo provenant d'une Webcam.

Chaque image extraite, ainsi que les interactions utilisateur (clic sur l'image, déplacement de la souris, texte saisi au clavier,…) sont ensuite envoyées au travers de tous les plugins activés (i.e. choisis et paramétrés par l'utilisateur). Il est de ce fait possible de développer des plugins complexes pour l'éditeur de SamGT, tel qu'un éditeur de vérité terrain, un plugin de traitement d'image ou d'extraction de sous-image,…

Tous ces modules sont bien sûr normalisés et héritent tous d'un même module parent (concept de l'héritage en programmation), permettant de bien définir les fonctionnalités que doit implémenter chaque composant et notamment des fonctionnalités d'intégration graphique dans l'éditeur, un plugin pouvant donc définir ensuite ses propres fenêtres et paramètres visuels.

Il a ainsi été possible par exemple, d'encapsuler facilement les appels des méthodes des fonctions de reconnaissance de la bibliothèque OpenCV (Figure 175), de créer des plugins à affichage complexe, comme l'édition de diagramme de traitement d'image (Figure 176) manipulant ses propres plugins (Figure 177) ; et bien sûr, ce qui nous intéresse ici, la création de fichiers de vérité terrain (Figure 178).

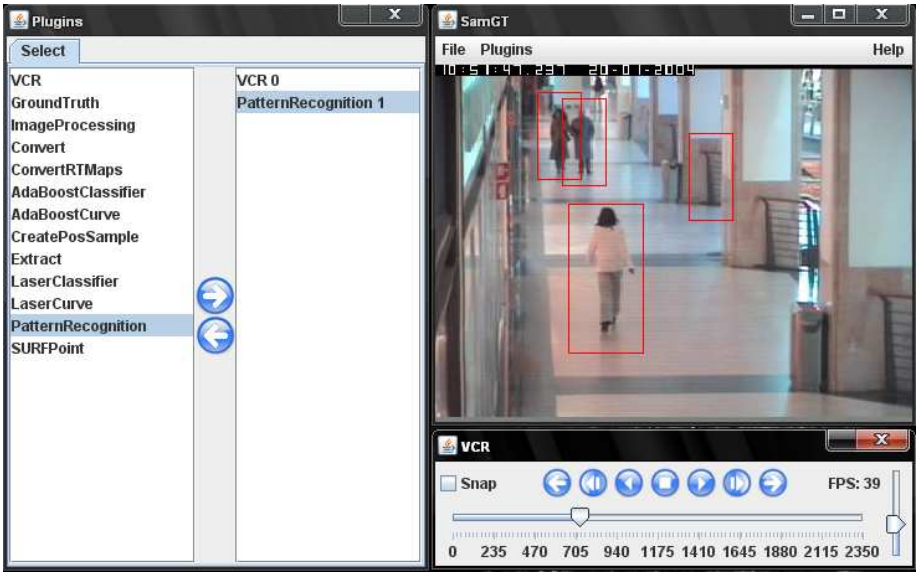


Figure 175 - Plugin de reconnaissance d'image OpenCV, sous SamGT

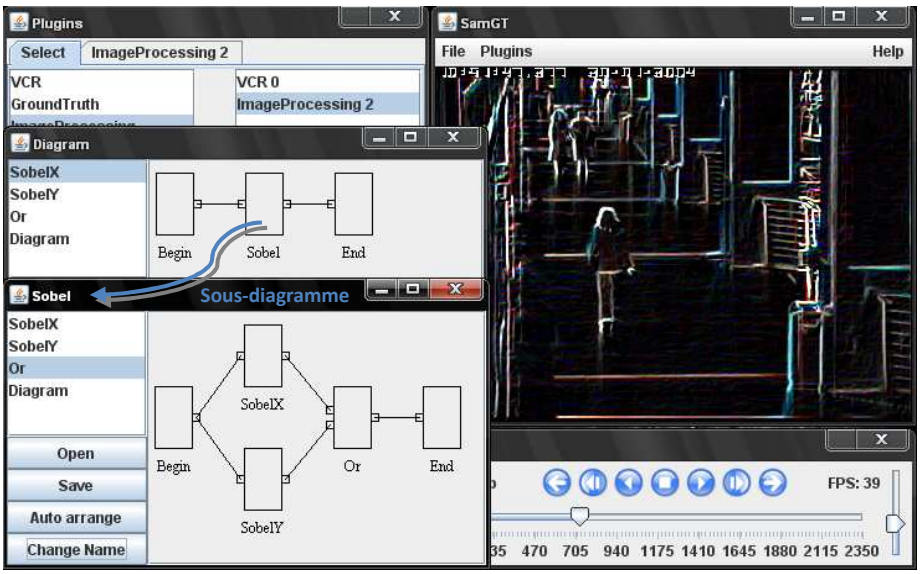


Figure 176 - Plugin de traitement d'image sous forme de diagramme sous SamGT

```
public class Or extends Plugin {
    public Object[] inputs  = {BufferedImage.class, BufferedImage.class};
    public Object[] outputs = {BufferedImage.class};

    public void process() {
        setOutput(0, Functions.or((BufferedImage)inputs[0], (BufferedImage)inputs[1]));
    }
}
```

Typage des entrées / sorties

Figure 177 - Définition d'un plugin de traitement d'image sous SamGT

Figure 178 - Création d'une vérité terrain sous SamGT

b) *Le format de fichier*

Le format des fichiers de vérité terrain générés (d'extension « .gt ») a été défini afin de pouvoir être facilement lisible par un opérateur humain ou par une machine, et d'être évolutif (i.e. supporter l'ajout de nouveaux types ou sous-types d'objet détectables sans détériorer les anciens « formats »). Ainsi comme le montre l'Exemple 4, un fichier GT est un fichier texte dont chaque ligne correspond à un objet détectable dans une image. Par exemple, si 2 panneaux apparaissent sur une seule image, 2 lignes seront présentes dans le fichier, chacune d'elle correspondant à l'un des panneaux.

Cet exemple montre aussi que les données sont formatées de façon très stricte dans le fichier, permettant une lecture logicielle aisée.

```
Timestamp / frameindex x1 y1 x2 y2 id 'type' 'subtype'
00:00.0000 / 0 179 177 197 195 0 'Speed Limit Signs' '110'
00:00.0000 / 0 467 169 486 188 1 'Speed Limit Signs' '50'
00:00.0250 / 1 179 177 197 195 0 'Speed Limit Signs' '110'
00:00.0250 / 1 468 170 487 189 1 'Speed Limit Signs' '50'
00:00.0500 / 2 177 177 195 195 0 'Speed Limit Signs' '110'
00:00.0500 / 2 470 170 489 189 1 'Speed Limit Signs' '50'
00:00.0750 / 3 173 177 191 195 0 'Speed Limit Signs' '110'
```

Exemple 4 – Exemple de fichier GT

Chaque ligne de ce fichier comporte les informations nécessaires pour l'évaluation des algorithmes sous rtMaps et la relecture du fichier dans l'éditeur :

- le timestamp de l'image (i.e. sa datation), nécessaire pour la relecture sous rtMaps
- l'index de la frame, pour la relecture dans l'éditeur SamGT
- la position et la taille de l'objet, pour apparier avec les résultats des algorithmes
- l'identifiant unique de l'objet, permettant de le suivre temporellement dans le flux vidéo et de le différencier des autres objets.
- le type et le sous-type de l'objet à détecter, un fichier de vérité terrain pouvant contenir plusieurs types d'objets (panneau 110, panneau 50, véhicule, feu tricolore,…). Ces deux dernières informations sont écrites en toutes lettres, il sera donc possible d'ajouter de nouveaux types sans aucun problème.

L'un des problèmes pouvant découler de la création de fichier GT par différentes personnes pour une même séquence, est de pouvoir les rassembler en un seul fichier et de les partager avec tous. Le cas idéal serait que tous les membres du laboratoire (ceux travaillant sur les problématiques ADAS de détection) travaillent sur une même séquence de test et que, celui ayant créé une vérité terrain pour les panneaux de limitation de vitesse par exemple, puisse la fusionner avec celle de celui qui aurait travaillé sur la détection des feux tricolores ou des piétons ou des véhicules. Ceci est rendu possible via le site Sam où il est facile d'ajouter un fichier GT à une séquence, un algorithme sur le site se chargeant de fusionner cette nouvelle vérité terrain avec celle existante déjà sur le serveur (Figure 179).

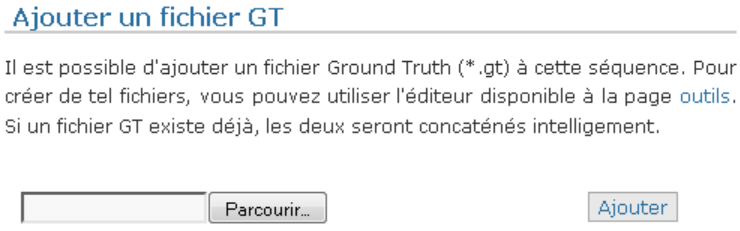


Figure 179 - Partage des fichiers de vérité terrain sur le serveur SAM

5.2.3. Evaluation des algorithmes

a) Fonctionnement

L'évaluation des algorithmes avec SamGT est très simple : il suffit de charger sous la même plateforme de prototypage ([rt]Maps) avec laquelle les algorithmes sont développés, le paquet de modules qui vont servir à comparer en temps réel les résultats de l'algorithme avec le fichier de vérité terrain.

Ainsi, il suffit, comme le montre la Figure 180, de :

- placer le module « Decoder » paramétré avec le fichier de vérité terrain et des types et sous-type des cibles à détecter, permettant par exemple de n'extraire que les panneaux de limitation de vitesse du fichier de vérité terrain ;
- de relier la sortie de ce module (i.e. les cibles *ground truth*) ainsi que la sortie de l'algorithme à évaluer (composé d'un ou plusieurs modules) à un module d'appariement (« *Matching* »), qui peut être, soit temporel, soit frame par frame fournissant le nombre de bonnes détections (*vrais positifs*), de non détections (*faux négatifs*) et de mauvaises détections (*faux positifs*) ;

- de relier ces statistiques à un module de comptage pour calculer les statistiques globales des performances de l'algorithme sur l'ensemble de la séquence de test ; et de les enregistrer dynamiquement dans un fichier Excel (version 2003) ou de les visualiser directement sous [rt]Maps.

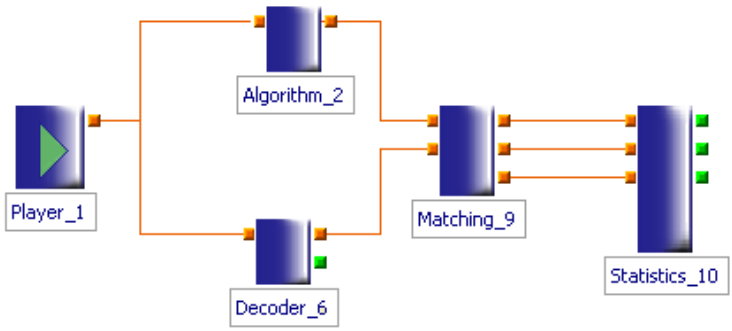


Figure 180 - Diagramme d'utilisation de SamGT

b) Appariement des cibles

L'appariement des cibles issues de l'algorithme de détection et de celle de la vérité terrain est l'étape la plus importante dans SamGT. En effet, un mauvais appariement fausserait ou embellirait les résultats de l'algorithme développé. Ainsi ce module, après de nombreuses réflexions, propose plusieurs paramètres afin de répondre aux divers cas de figure pouvant entrer en jeu dans l'évaluation d'une détection.

Tout d'abord, la question fondamentale à se poser est de savoir comment considérer que deux cibles (celle de l'algorithme et celle du fichier de vérité terrain) correspondent à la même détection ? En effet, du fait de la discrétisation d'une image et des conditions environnementales (phénomène d'éblouissement de la caméra, arrière plan de l'objet dans l'image,…), il est impossible, même pour un opérateur humain d'encadrer correctement un objet dans une image. Dans l'exemple de la Figure 181, il est très difficile de savoir avec précision quelle est la limite physique du bord droit du panneau de limitation de vitesse. Certes il est possible d'appliquer des contraintes géométriques sur la taille des objets à détecter, afin d'aider le processus d'encadrement (contraintes incluses dans l'éditeur de SamGT), mais celui-ci restera, quoi qu'il arrive, imparfait.

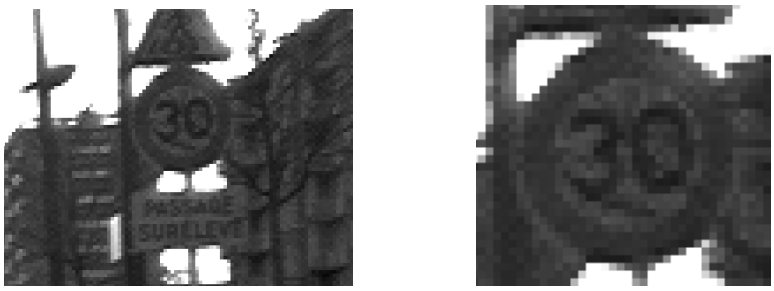


Figure 181 - Un panneau de limitation de vitesse de 30km/h (à gauche), la même image zoomée (à droite)

Ceci pour expliquer qu'une simple comparaison géométrique (taille et position dans l'image) n'est pas suffisante. Les deux cibles ne pouvant quasiment jamais être identiques de ce point de vue. De plus pour diverses autres raisons, un algorithme de détection peut fournir une cible plus petite (par exemple, dans le cas des détections des panneaux de limitation de vitesse, si il détecte le cercle intérieur du panneau et non l'extérieur) et / ou, plus ou moins décalée par rapport à la réalité (i.e. par rapport au fichier de vérité terrain), ainsi que le montre la Figure 182.

Figure 182 - Différentes détections possibles pour le panneau de limitation de vitesse par un algorithme (en vert) comparé à la vérité terrain (en rouge) bien que conservant un ratio de 1.

Ce problème d'appariement de cibles détectées n'est pas nouveau. Ainsi dans ses travaux sur la détection visuelle de véhicule, (Khammari, 2006) définit deux critères de mesure de similarité (Figure

183) : S_1 correspondant à l'aire de l'intersection divisée par l'aire de l'union des deux cibles (Figure 184); et S_2 qui est l'aire de l'intersection divisée par l'aire de la plus petite des deux cibles.

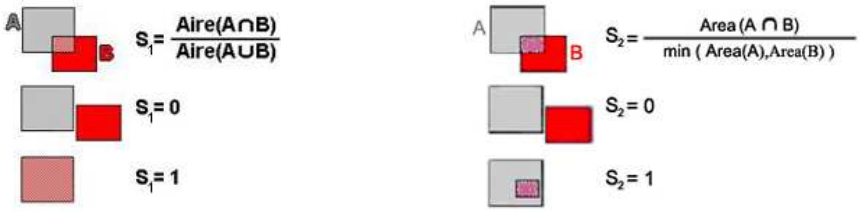


Figure 183 - Différents critères de similarité entre 2 cibles dans une image

La principale différence entre ces deux critères réside dans le fait que S_2 est plus tolérant vis à vis de l'écart entre tailles des cibles à comparer et est donc plus adapté à l'appariement de cibles à des instants différents quand l'objet à détecter, ou la caméra, est en mouvement.

La problématique ici est l'appariement de deux cibles (détections) correspondant au même instant. Ainsi le premier critère (S_1) répond au mieux à notre problème. Ce critère a donc été retenu, et un premier paramètre de seuil est à définir par l'utilisateur, correspondant donc au pourcentage de chevauchement toléré des deux cibles. Plus ce paramètre sera faible (proche de 0), plus l'appariement sera tolérant, et à l'inverse, plus ce paramètre sera important (proche de 1), plus l'appariement sera strict. Une valeur de 1 correspondant à un appariement parfait. La valeur de 0,75 est généralement employée pour tester nos algorithmes.

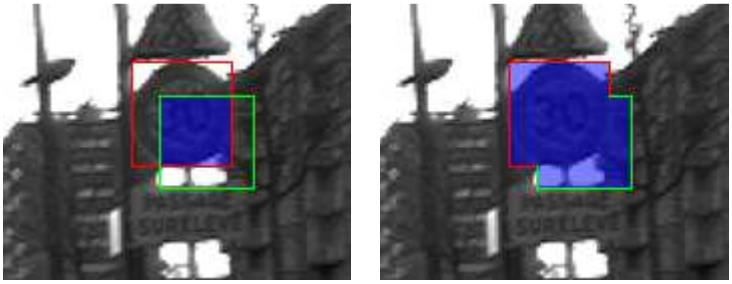


Figure 184 - Intersection des deux cibles (à gauche) et leur union (à droite)

Une autre question qui se pose est de savoir comment interpréter les résultats d'un algorithme détectant plusieurs fois le même objet dans une image. En effet, si un algorithme de détection peut se tromper sur la position et la taille de l'objet détecté, il peut aussi cumuler ses erreurs et détecter plusieurs fois cet objet sur une même image, comme le montre par exemple la Figure 185. Faut-il considérer toutes ces cibles comme une seule et bonne détection (si elles répondent toutes au critère de chevauchement), ou n'en considérer qu'une de correcte et les autres en tant que mauvaises détections ? La réponse à cette question est laissée au choix de l'utilisateur au travers d'un paramètre du module d'appariement, le choix changeant grandement les « résultats » de l'algorithme (i.e. le nombre de vrais positifs et de faux positifs,…).

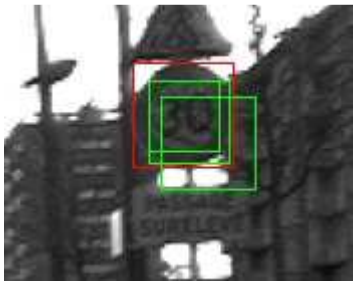


Figure 185 - Plusieurs détections possibles au même instant par un algorithme de détection (en vert), comparé à la vérité terrain (en rouge)

c) *Statistiques générées*

Le module de statistique (celui qui effectue le comptage du nombre de vrais positifs, faux positifs,… issus du module d'appariement) génère dynamiquement deux fichiers de statistique :

- L'un au format Excel 2007 (*xml*) présentant des statistiques dans une dizaine de feuilles Excel.
- L'autre au format texte contient les données tabulées (qui sont les données brutes, aussi présentes dans le fichier Excel) facilement exploitable par un logiciel tiers ou si l'utilisateur désire développer son propre programme de traitement.

La première feuille du fichier Excel généré contient les statistiques brutes : un tableau représentant les données ligne par ligne pour chacune des détections, ou non détections, en précisant les informations nécessaires à leurs exploitation et interprétation :

- Le type de détection (vrai positif, faux positif ou faux négatif)
- Le type et sous-type de l'objet
- Diverses propriétés (position, taille, identifiant, timestamp,…)

Les autres pages du fichier fournissent des statistiques issues du croisement des données brutes, pour faciliter l'interprétation des résultats, ce qui est nécessaire pour l'étude des améliorations à apporter à ses algorithmes (i.e. savoir sur quel point doivent porter les efforts futurs). Par exemple sont calculés les statistiques sur les détections par rapport à :

- la taille des objets : il est ainsi possible de savoir par exemple à partir de quelle taille dans l'image l'algorithme arrive à détecter systématiquement les objets ;
- leur type et sous-type : savoir si certains objets sont mieux reconnus que d'autre (par exemple les panneaux à 3 chiffres par rapport aux panneaux à 2 chiffres)
- leur identifiant (savoir que le $n^{ième}$ objet n'est que très mal reconnu par rapport aux autres et de pouvoir directement visualiser dans la vidéo comment celui-ci se présente, si il est à moitié caché par exemple)
- …

Cependant, même si ces statistiques générées semblent idéales pour l'étude des algorithmes de détection, elles amènent quand même à se poser quelques questions et notamment sur leur interprétation en cas de détection temporelle et non image par image. En effet, que veulent dire par exemple, les statistiques sur la détection par rapport à la taille des objets d'un point de vue temporel ? En effet, dans un processus de détection temporelle, la seule chose qui peu importer est d'avoir validé la présence de l'objet quand il est présent, que ce soit 50 mètres avant d'être passé à côté du véhicule, ou simplement 10 mètres ; et non pas à réussir à détecter cet objet successivement sur chacune des images. Là aussi, il faudra donc faire attention aux paramètres à étudier, mais SamGT est développé dans un but générique pour l'utilisation de tous pour tout type d'algorithme de détection.

d) Evaluation de jeux de paramètres

Mais l'utilisation de SamGT ne s'arrête pas là. Pour diverses raisons, il peut être nécessaire de tester un algorithme avec plusieurs paramètres possibles. Par exemple, l'utilisation d'un algorithme déjà existant oblige à trouver le bon jeu de paramètres à son exploitation pour répondre au mieux à un problème donné. Trouver ce bon jeu de paramètres peut être réellement fastidieux, même guidé par son expertise passée. Un autre cas de figure intéressant est de pouvoir évaluer son algorithme sous diverses conditions paramétrables en même temps ou, les unes à la suite des autres.

Dans ces deux cas, évaluer systématiquement son algorithme en changeant un à un les paramètres à tester et relever les résultats (i.e. le taux de bonnes détections, fausses détections,…) peut devenir très vite laborieux. En effet, dans le cas où le nombre de paramètres à tester n'est par exemple seulement que de 2, dont chacun peut avoir 10 valeurs différentes possibles, il faudrait évaluer l'algorithme 100 fois sur une séquence de test. Même si cette séquence est très courte, il faudrait un temps de travail assez conséquent pour relever tous les résultats (une séquence de 5min par exemple, demanderait environ 8h de test).

Pour palier à cette contrainte, SamGT propose un module ^{rt}Maps afin d'évaluer automatiquement des jeux de paramètres. Celui-ci, se charge d'exécuter autant de fois que nécessaire l'algorithme, et d'enregistrer les statistiques pour chaque jeu de paramètres testé. Il est ainsi possible de faire tourner ces nombreux tests en tâche de fond, ou de les exécuter la nuit, et de récupérer l'ensemble des résultats une l'évaluation terminée.

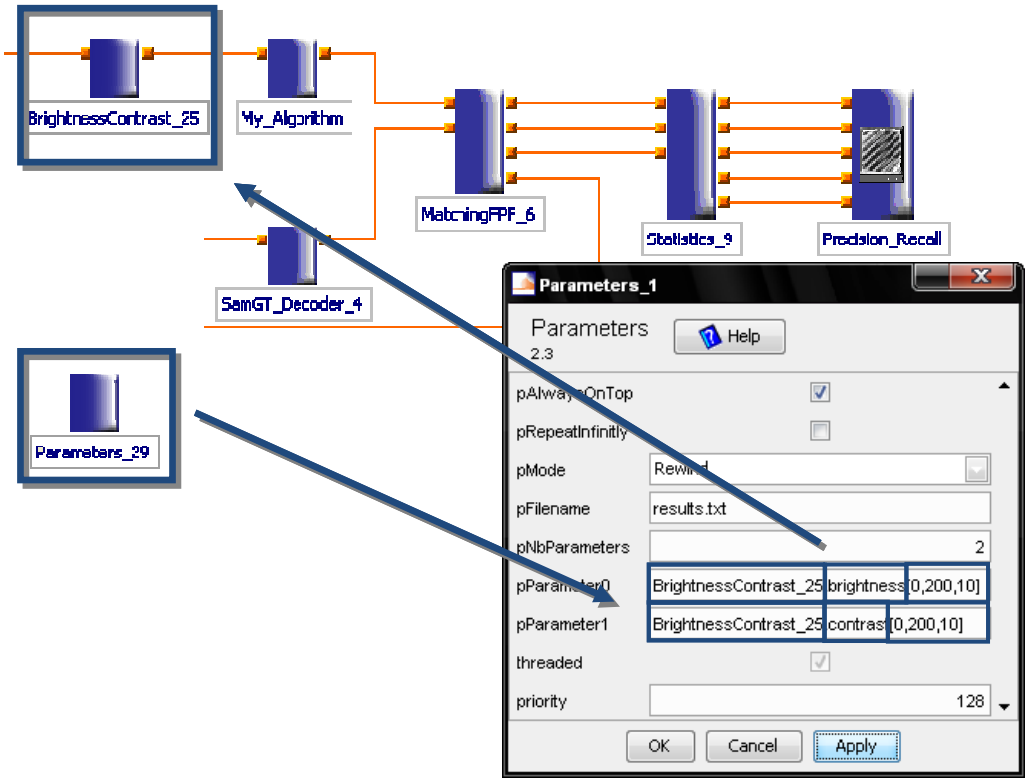


Figure 186 - Schéma de l'évaluation d'un jeu de paramètres sous rtMaps avec SamGT : ici les valeurs des propriétés *brightness* et *contrast* du module *BrigthnessContrast_25* en amont de l'algorithme à évaluer, sont automatiquement changées entre 0 et 200 avec un pas de 10.

Par exemple, dans le cas de l'évaluation de l'algorithme de reconnaissance des panneaux de limitation de vitesse, présenté au début de ce manuscrit, cet algorithme a été testé sous diverses conditions de luminosité et de contraste avec 20 valeurs possibles pour chacun de ces 2 paramètres. Soit 400 tests à effectuer sur la séquence de test. Bien entendu cette évaluation n'a pas été réalisée manuellement. Les résultats de ces 400 tests présentés Figure 87 ont été générés, à l'aide du diagramme schématisé par la Figure 186, où le module « *Parameters* » du package rtMaps de SamGT a permis de faire varier un à un ces deux paramètres.

L'avancement de la série de test ainsi qu'une heure estimée de la fin de l'évaluation de tous les jeux de paramètres est affichée dynamiquement dans une fenêtre de progression (Figure 89). Ceci permet à l'évaluateur de savoir quand récupérer ses résultats sans avoir à attendre (ce qui était l'un des buts de ce module).

A titre purement informatif, la Figure 187, présente le schéma rtMaps utilisé pour l'évaluation de l'algorithme de détection des panneaux de limitation de vitesse. Cette figure montre qu'il est tout aussi simple d'évaluer des paramètres sur un diagramme important (comportant plusieurs dizaines de modules).

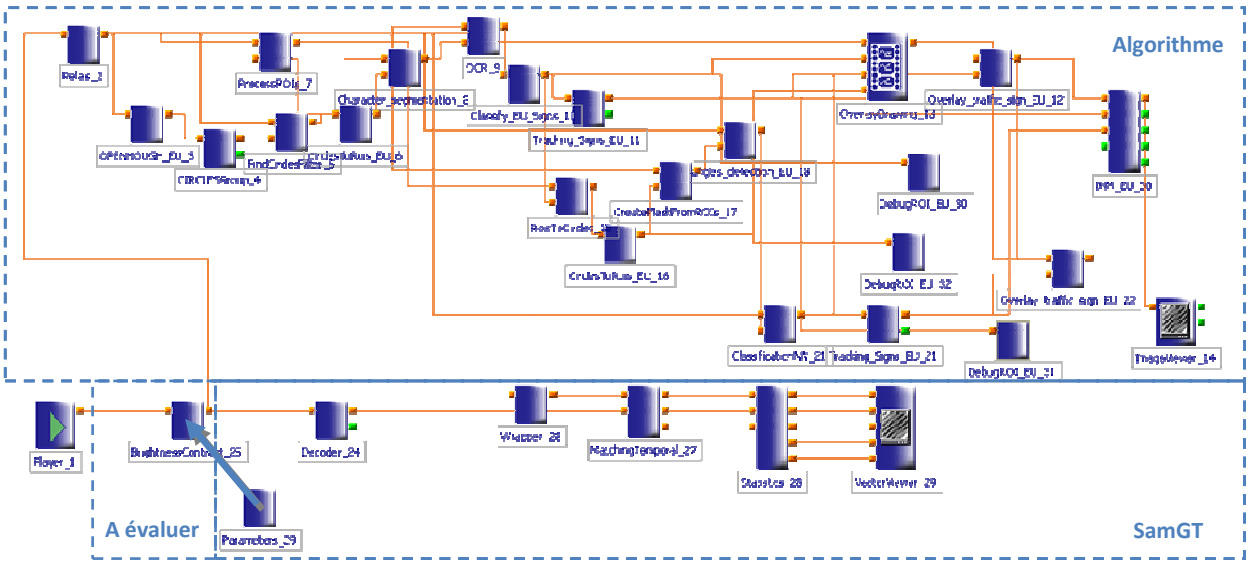


Figure 187 - Diagramme d'évaluation du jeu de paramètres luminosité / contraste pour de l'algorithme de détection des panneaux de limitation de vitesse présenté dans ce mémoire

5.3. Apprentissage numérique et classification d'image

5.3.1. Motivations

Enfin la dernière question soulevée dans ce chapitre est celle de l'amélioration des algorithmes d'apprentissage numérique par l'apport de nouvelles idées au sein d'un laboratoire de recherche. En effet, lors de la mise en place d'algorithmes de détection visuelle, il n'est pas rare d'utiliser de tels algorithmes d'apprentissage numérique ou tout simplement de porter ses travaux de recherche directement sur l'amélioration d'une méthode d'apprentissage.

Au sein même du centre de robotique, nous sommes nombreux à utiliser ces algorithmes (réseau de neurones, Adaboost,…) pour nos systèmes de détection visuelle (de panneaux, de voitures, de piétons,…), mais nous utilisions divers outils indépendamment des uns et des autres (Winseville pour Adaboost, Janet pour les réseaux de neurones,…). Même si certains de ces outils ont été développés puis améliorés en interne au laboratoire, notamment Winseville par (Abramson, 2005) qui vise à utiliser des algorithmes génétiques pour la génération de *classificateurs faibles* pour l'algorithme Adaboost ; amélioré par les travaux postérieur de (Ghorayeb, 2007) qui porta notamment ces algorithmes lourds en calcul sur carte graphique (bien plus puissante pour les calculs parallèles) ; il n'existait pas d'outil unifié et simple d'emploi pour l'utilisation de ces algorithmes.

Il existe certes quelques bibliothèques déjà existantes (OpenCV, Torch, LTI-Lib, Camellia, AForge,…) proposant un set d'algorithmes de classification / reconnaissance d'image, mais celles-ci ne sont pas vraiment adaptées à notre problématique. En effet, il est très difficile de rentrer dans le code source de ces bibliothèques. Celles-ci sont écrites en C/C++ afin de minimiser le temps d'exécution des algorithmes et elles cherchent souvent à pouvoir traiter n'importe quel type d'image (8bits, 12bits, 16bits, couleur, niveau de gris, avec des données alignées ou non,…) ce qui est, dans un contexte d'étudier et de développer des algorithmes de reconnaissance, pas vraiment intéressant. Le code source résultant est donc, en contre partie, plus ou moins complexe à appréhender et demande de nombreuses heures afin de pouvoir être complètement opérationnel dans l'ajout de fonctionnalités de ces bibliothèques.

L'idée du logiciel LeViS (LEarning algorithms in VIsion System) présenté dans cette section, vient donc du besoin de posséder un outil (une plateforme) simple et unifié au sein du laboratoire regroupant tous ces algorithmes couramment utilisés que chacun pourrait employer et améliorer. L'originalité de ce logiciel est donc inscrite dans ces buts fondamentaux qui sont de pouvoir :

- comparer aisément les performances de chacune des méthodes de reconnaissance sur diverses bases d'apprentissage ;
- utiliser facilement et rapidement au sein de nos applications un algorithme de détection ;
- améliorer facilement ces algorithmes via de nouvelles idées (en pouvant accéder et rééditer le code source simplement et aisément) ;
- créer de nouveaux algorithmes rapidement sans se soucier des problèmes de gestion de bases de données d'images pour l'apprentissage / les tests de ce nouvel algorithme et bien entendu le comparer aux autres déjà existants.

5.3.2. Présentation technique

a) Présentation générale

L'outil LeViS est écrit en Java pour la grande lisibilité du code de ce langage et les grandes facilités qu'il apporte : reconnaissance native des formats d'images bmp, png, gif,… couramment utilisés dans les bases d'apprentissage d'image ; sa syntaxe proche du C++ utilisé dans nos développement sous [rt]Maps ; son ramasse miette (*garbage collector*) qui permet d'éviter les fuites mémoire en permettant au développeur de gagner du temps et de se focaliser sur son application ; la possibilité de création d'interface graphique native de ce langage (les composant graphique Swing) ; ainsi que sa compatibilité multiplateforme (Windows, Linux, 32bits, 64bits,…).

Comme le montre le schéma d'implémentation du logiciel Figure 188, un effort a été mené pour bien dissocier d'un point de vue conceptuel, ainsi que dans le code source du logiciel, les codes des algorithmes d'apprentissage, des algorithmes de classification, ainsi que de ceux de l'interface graphique.

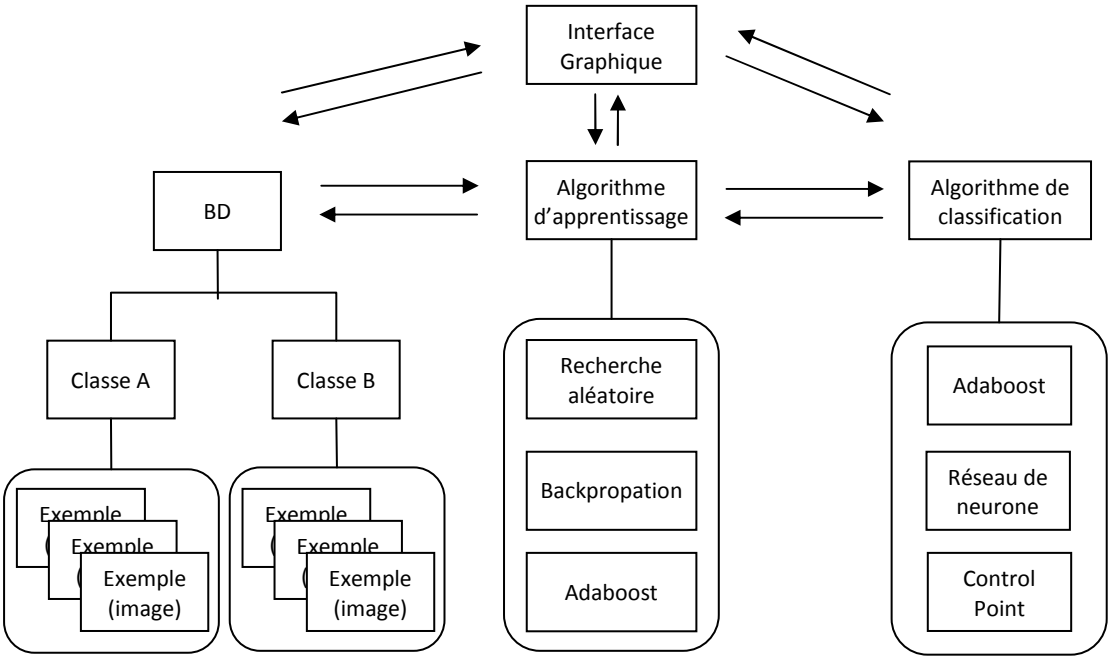


Figure 188 - Schéma conceptuel de LeViS

LeViS se ramène donc dans sa conception à la définition même d'un algorithme de classification et d'un algorithme d'apprentissage (Figure 189) :

- Un algorithme de classification doit être vu comme une boite noire paramétrable qui, lorsqu'une image lui est présentée, fournit la classe de cette image (i.e. piéton, voiture, moto, panneau de limitation de vitesse,…);

- Un algorithme d'apprentissage doit, par présentation successive d'images à un algorithme de classification, trouver le bon jeu de paramètres de cette boite noire afin que celle-ci se trompe le moins possible dans la classification des images présentées.

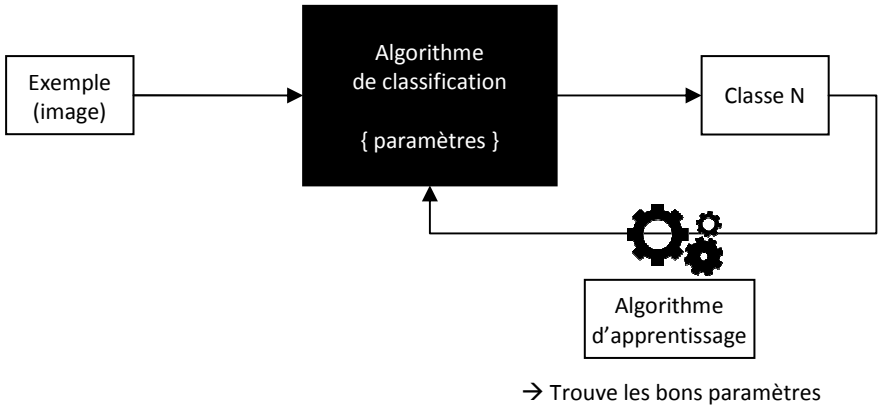


Figure 189 - Définition d'un algorithme d'apprentissage et d'un algorithme de classification

Certains algorithmes d'apprentissage sont propres à un algorithme de classification (par exemple la Backpropagation pour les Réseaux de neurones,…), mais d'autre sont génériques et peuvent être utilisés pour n'importe quel algorithme de classification. Ainsi, si un algorithme de classification implémente certaines fonctionnalités prédéfinies par l'un des algorithmes d'apprentissage (le concept d'*interface* du langage Java), il pourra automatiquement être utilisé avec celui-ci. Par exemple, l'implémentation d'une fonction « *randomize* » au sein d'un tel algorithme (fonction qui doit affecter des valeurs aléatoires aux paramètres), permet d'utiliser l'algorithme d'apprentissage de recherche aléatoire avec celui-ci.

Ce même concept permet aux algorithmes d'apprentissage et de classification, d'implémenter des fonctionnalités prédéfinies pour l'affichage graphique. Ainsi leurs divers paramètres sont automatiquement affichés et gérés par l'interface graphique, l'utilisateur pourra alors aisément choisir et paramétrer les algorithmes qu'il veut utiliser.

Avec une telle normalisation, il est possible d'implémenter chacun des algorithmes au sein de LeViS de façon indépendante (ce qui facilite leur programmation et la lisibilité du code); ainsi que de réutiliser directement les codes des algorithmes de classification (munis des fichiers de paramètres appris) dans d'autres logiciels.

Enfin l'utilisation du langage Java permet d'ajouter une « surcouche » pour utiliser les algorithmes d'apprentissage de façon distribuée sur plusieurs ordinateurs en réseau. En effet, le langage Java intègre depuis l'une de ses premières versions, la 1.1 en 1997, un moteur RMI – Remote Method Invocation. Sans entrer dans les détails, RMI est un ensemble de classes du langage Java permettant de manipuler des objets sur des machines distantes de manière similaire aux objets sur la machine locale sans se soucier des communications réseaux. Ainsi, un algorithme gourmand en temps de calcul peut être exporté facilement sur chacun des ordinateurs du réseau et être « multi-threadé » sur chacun des cœurs du processeur de chaque machine.

Tous les algorithmes de classification ont été de façon similaire (i.e. du code source indépendant) réécrits en langage C++ de façon à pouvoir être compilés au sein d'un package ^{rt}Maps pour l'utilisation dans nos algorithmes de détection, et au sein d'un logiciel de test. Ce dernier permet de

comparer les résultats obtenus avec l'implémentation des algorithmes de classification en Java et celle en C++ sur une même base de test afin de vérifier que les deux implémentations fournissent les mêmes résultats. Une section dans le logiciel LeViS, où il suffit d'indiquer l'emplacement de ce logiciel (fichier exécutable) ainsi que l'emplacement de la base de test, permet d'automatiser ce processus. Il est alors possible de réécrire les algorithmes de classification dans un autre langage de programmation (Python, Caml,…), de les compiler dans un programme de test, et d'utiliser l'outil dédié dans LeViS pour valider leur implémentation.

b) *Exemple d'implémentation d'algorithmes d'apprentissage et de classification*

Voici deux exemples montrant la simplicité d'écriture d'algorithmes d'apprentissage et de classification au sein de LeViS.

L'algorithme de classification Exemple 5 est très simple : il compare (fonction *classify*) la valeur de deux pixels dans l'image. Si la différence est supérieure à un seuil alors l'exemple est classé positif, sinon négatif (c'est donc un algorithme de classification binaire). Les coordonnées des deux pixels à comparer ainsi que le seuil sont les paramètres de cet algorithme. Ces paramètres peuvent être tirés aléatoirement via la fonction *randomize* que l'algorithme est obligé de définir vu qu'il implémente l'interface *IRandom* (définie plus loin).

```
public class DemoClassifier extends Classifier implements IRandom {

	private Point p1 = new Point();
	private Point p2 = new Point();
	private int threshold;
	private int maxThreshold = 255;

	public int classify(Sample sample) {
		int pix1 = sample.getPixel(p1);
		int pix2 = sample.getPixel(p2);
		return ((pix1 - pix2) > threshold) ? Classifier.POSITIVE : Classifier.NEGATIVE;
	}

	public void randomize() {
		p1.x = SRandom.getInstance().nextInt(_model.width);
		p2.x = SRandom.getInstance().nextInt(_model.width);
		p1.y = SRandom.getInstance().nextInt(_model.height);
		p2.y = SRandom.getInstance().nextInt(_model.height);
		threshold = SRandom.getInstance().nextInt(maxThreshold);
	}
}
```

Exemple 5 - Exemple d'un algorithme de classification comparant 2 points dans une image

La création d'un algorithme d'apprentissage est tout aussi simple. Par exemple, un algorithme d'apprentissage stochastique afin de trouver le bon jeu de paramètres d'un algorithme de classification est décrit par le code de l'Exemple 6. Dans cet exemple, la fonction d'apprentissage (la fonction *learn*) itère sans fin en créant à chaque tour un nouvel algorithme d'apprentissage du même type que celui en paramètre. Les paramètres de cet algorithme de classification sont tout d'abord initialisés aléatoirement (fonction *randomize* vue plus haut). Puis est calculée, de façon complètement transparente car utilisant des fonctions implémentées nativement au sein de LeViS, l'erreur de classification de cet algorithme sur la base d'apprentissage (fonction *computeLearningError*). Si cette erreur est plus faible que l'erreur du meilleur algorithme de classification déjà trouvé jusqu'ici, alors celui-ci est sauvegardé. Enfin, l'appel à la fonction *learnStep* de LeViS, permet de mettre automatiquement à jour le graphe d'apprentissage et de calculer la condition d'arrêt (typiquement si le nouveau taux d'erreur d'apprentissage est inférieur à un seuil).

```
public class DemoLearner extends Learner {

	public interface IRandom {
		public void randomize();
	}

	private Classifier classifierModel = new DemoClassifier();

	protected Classifier learn(DataBase learnData) throws Exception {
		Model sampleModel = learnData.getSampleModel();
		Classifier bestClassifier = null;
		double error = 1;
		while (_learnLoop) {
			Classifier classifier = classifierModel.getNewInstance(sampleModel);
			((IRandom)classifier).randomize();
			classifier.computeLearningError(learnData);
			if (classifier.learningError < error) {
				bestClassifier = classifier;
				error = classifier.learningError;
			}
			learnStep(bestClassifier);
		}
		addfoundClassifier(bestClassifier);
		return bestClassifier;
	}
}
```

Exemple 6 - Exemple d'un algorithme d'apprentissage aléatoire

Ces deux codes intégrés dans LeViS seront directement affichés dans l'interface graphique et utilisables. Il est donc clair que créer un nouvel algorithme d'apprentissage ou de classification, ajouter de nouvelles idées au sein du code d'un algorithme,… est une chose réellement aisée ce qui était l'une des motivations de la création du logiciel LeViS.

5.3.3. Algorithmes implémentés

Sont présentés ici de façon succincte les algorithmes d'apprentissage et de classification déjà implémentés au sein de LeViS qui étaient couramment utilisés au laboratoire. Ce paragraphe n'a pas pour but de constituer un état de l'art des systèmes d'apprentissage numérique pour la reconnaissance d'image, ni même d'être une étude comparative de ces différentes techniques. Le lecteur pourra se reporter aux références citées dans chacun des sous-paragraphes suivants afin de compléter sa lecture.

a) Réseau de neurones

Un réseau de neurones artificiel est un modèle de calcul dont la conception est très schématiquement inspirée du fonctionnement des vrais neurones. Ce réseau calculatoire est en général composé d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente (Figure 190).

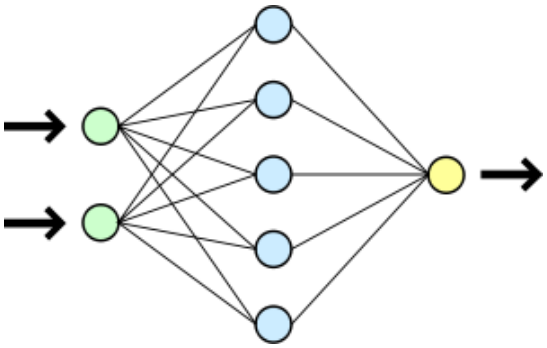


Figure 190 - Vue simplifiée d'un réseau artificiel de neurones

Chaque couche (i) est composée de N_i neurones, prenant leurs entrées sur les N_{i-1} neurones de la couche précédente. À chaque synapse est associé un poids synaptique, de sorte que les N_{i-1} valeurs sont multipliées par ce poids, puis additionnées par les neurones de niveau i. Les valeurs calculées sont ensuite utilisées dans une fonction d'activation qui établira la force du signal (la valeur) envoyée par le neurone courant de niveau i (Figure 191). La fonction d'activation sert à introduire une non-linéarité dans le fonctionnement du neurone. Des exemples classiques de fonctions d'activation sont les fonctions : sigmoïde et tangente hyperbolique.

Au-delà de cette structure simple, le réseau de neurones peut également contenir des boucles qui en changent radicalement les possibilités mais aussi la complexité. De la même façon que des boucles peuvent transformer une logique combinatoire en logique séquentielle, les boucles dans un réseau de neurones transforment un simple dispositif de reconnaissance en une machine complexe capable de toutes sortes de comportements (Wikipedia, 2009).

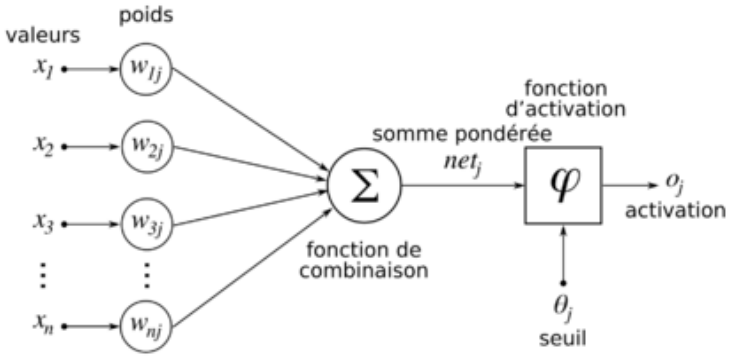


Figure 191 - Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

La technique la plus populaire pour l'apprentissage (i.e. trouver les poids optimum des connexions des neurones) de tels réseaux est la rétropropagation du gradient (plus communément appelée par son terme anglais "*backpropagation*") : les valeurs calculées en sortie sont comparées avec les valeurs attendues pour calculer une fonction d'erreur prédéfinie. Par différentes techniques, cette fonction d'erreur est utilisée afin d'ajuster les poids des connexions des neurones dans le but de réduire cette erreur. Après avoir répété cette opération sur un nombre de cycles d'apprentissage assez important, le réseau converge habituellement vers un état où l'erreur calculée est faible.

(Cybenko, 1989) a émis et prouvé le théorème qu'un perceptron multicouche (un réseau de neurones sans boucle dont les connexions ne se font qu'avec les neurones de la couche précédentes, du type de la Figure 190) d'une seule et unique couche cachée (i.e. qui comporte une couche d'entrée, une couche intermédiaire dite cachée et une couche de sortie) est capable d'approximer n'importe quelle fonction continue à multiple variables et ce à n'importe quel degré de précision si les variables du réseau (poids, fonction de transfert, nombre de neurones de la couche cachée) sont bien choisies.

Le problème étant alors de choisir le bon nombre de neurones de la couche cachée (l'algorithme d'apprentissage de Backpropagation trouvant les poids des connexions).

Le lecteur désireux d'approfondir le sujet peut se reporter à (Kröse, et al., 1996) qui introduit d'une façon très claire toute la théorie mathématique de l'apprentissage pour les différentes topologies des réseaux de neurones : des simples perceptrons (Adaline, MLP,…), aux réseaux récurrents (Elman, Jordan, Hopfield,…), en passant par les réseaux auto associatifs (Kohonen, Art1, Art2,…) et par les réseaux stochastiques (Boltzmann).

LeViS intègre une forme simple des réseaux des neurones en tant qu'algorithme de classification: ceux de type Perceptron multicouche (chaque neurone d'une couche est connecté à chaque neurone de la couche précédente, sans boucle) dont les valeurs des neurones d'entrée représentent les valeurs des pixels d'une image à classifier; muni de l'algorithme d'apprentissage classique de rétropropagation du gradient.

b) Adaboost

L'idée du boosting réside dans la question suivante : est-il possible de créer un algorithme d'apprentissage efficace (*strong learner*) à partir d'un ensemble d'algorithmes peu efficaces (*weak learner*) ?

Un algorithme peu efficace étant un algorithme un peu meilleur que le hasard (i.e. dont le taux d'erreur est un peu moins que 0.5). Au contraire, un algorithme efficace est un algorithme dont le taux d'erreur est proche de 0.

Il existe de nombreux algorithmes de boosting : Adaboost, GentleBoost, RankBoost, LPBoost,… Adaboost fut l'une des premières méthodes pleinement fonctionnelles permettant de mettre en œuvre le principe de boosting et a été appliqué avec succès à la reconnaissance de visage dans des vidéo en temps réel (Viola, et al., 2001).

Le principe de base de l'algorithme Adaboost (Adaptive Boosting) est de trouver un à un, d'une manière itérative, un ensemble de classificateurs faibles pour créer un classificateur fort. Chaque classificateur faible est appris sur la même base d'apprentissage exception faite que les poids des exemples sont pondérés en fonction de leur difficulté à être correctement classés par le classificateur fort courant. Ainsi, initialement tous les exemples ont un poids identique, puis à chaque étape, les poids des exemples mal classés par le classificateur fort sont augmentés. Ceci force le classificateur faible courant (en train d'être appris) à se concentrer sur les exemples difficiles.

Adaboost est adaptatif, dans le sens où les classificateurs faibles sont affublés d'un poids correspondant à leur efficacité (i.e. à leur taux d'erreur de classification de la base en cours, qui contient donc des exemples à probabilités différentes). Ces poids permettent d'effectuer un vote pondéré afin de décider de la classe finale de l'objet à reconnaitre. Vers la fin de l'apprentissage, le poids des exemples difficiles à apprendre devient largement dominant. Si on peut trouver un classificateur faible performant sur ces exemples, il sera alors doté d'un coefficient considérable. L'une des conséquences possibles est que les exemples bruités, sur lesquels finit par se concentrer l'algorithme, perturbent l'apprentissage.

(Freund, et al., 1999) qui sont à l'origine de l'algorithme Adaboost, ont prouvé que si chaque classificateur élémentaire assemblé a une erreur inférieure à 0.5, alors l'erreur empirique du classificateur fort décroit exponentiellement avec le nombre d'étapes.

Malheureusement l'algorithme d'Adaboost n'est applicable que pour la création de classificateur binaire, ce qui le rend inutilisable pour le problème de classification des panneaux de limitation de vitesse. Mais des travaux de recherche en cours au sein du laboratoire visent à rendre cet algorithme multi-classes.

La Figure 192 montre un exemple simple d'application d'Adaboost pour classifier des données. La taille de chaque exemple (+ ou -) est proportionnelle à son poids.

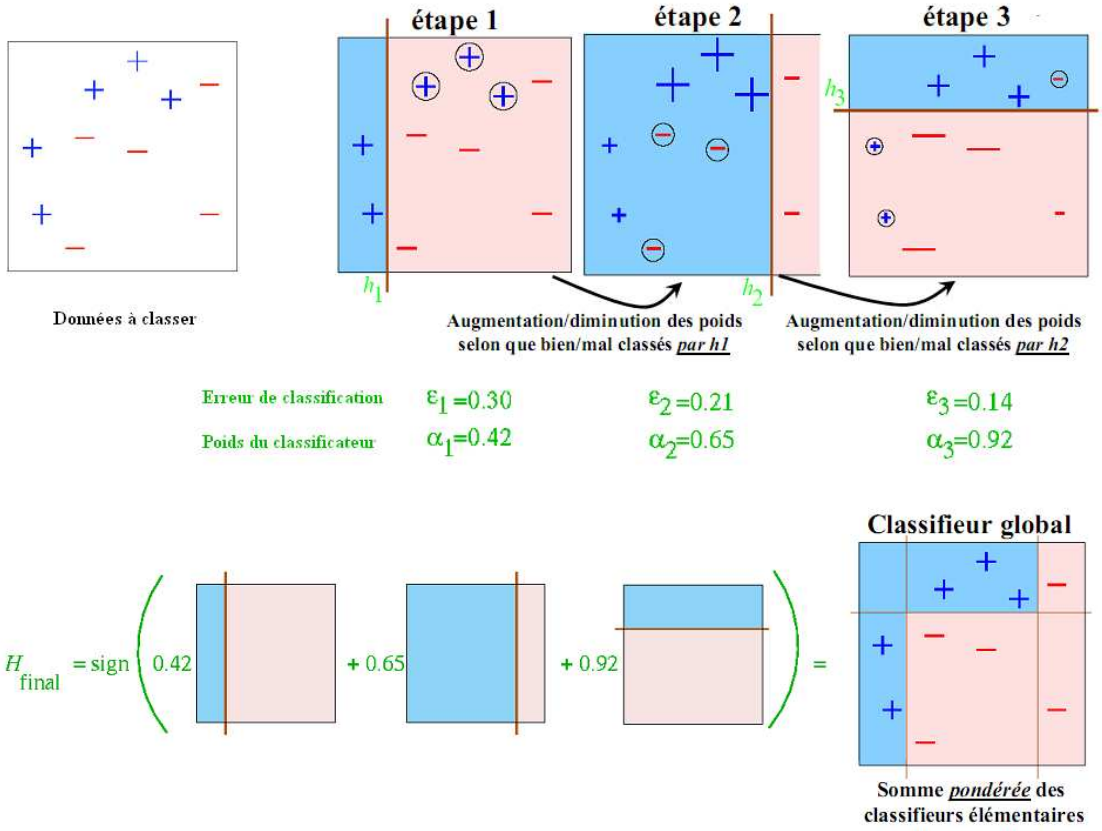


Figure 192 - Exemple d'application de l'algorithme Adaboost

Dans le domaine de la reconnaissance visuelle d'objet, les classificateurs faibles couramment utilisés sont ceux introduits par (Viola, et al., 2001) : les classificateurs de Haar. Ces classificateurs calculent la différence absolue entre la somme des valeurs des pixels en bleu et en blanc. Si cette différence est supérieure à un seuil, alors l'exemple est classé positivement (Figure 193).

Figure 193 - Exemple de classificateurs de Haar

D'autres classificateurs faibles ont été proposés pour Adaboost comme les Control-Points (Abramson, et al., 2005) qui sont plus rapides à calculer et plus robustes aux variations d'illumination (Figure 195). Ils sont constitués de deux groupes de points (positifs et négatifs). Un exemple est classé positif, si la différence entre la valeur minimale représentée par un pixel positif et la valeur maximale d'un pixel négatif (et inversement) est supérieur à un seuil (Figure 194).

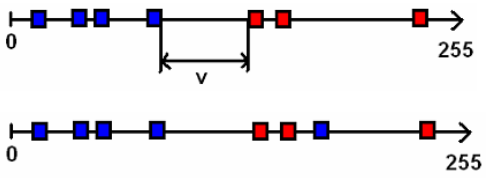


Figure 194 - Représentation linéaire d'un classificateur de type Control-Point : en haut exemple positif respectant le seuil V - en bas exemple négatif

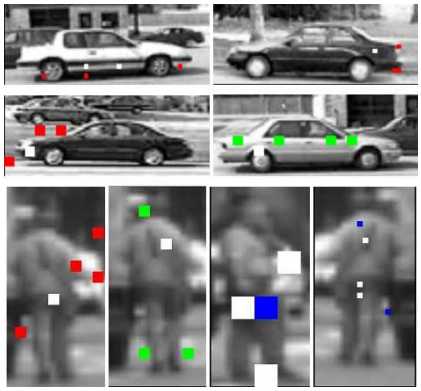


Figure 195 - Exemple de classificateur Control-Points

Une amélioration des Control-Points, les Connected-Control-Points, a été apporté par (Stanciulescu, et al., 2007) en ajoutant une contrainte de connexité sur un voisinage de 8, ce qui implique que chaque point doit toucher au moins un autre point par un coin (Figure 196). Ils ont prouvé que l'ajout de cette contrainte permet d'améliorer significativement l'algorithme.

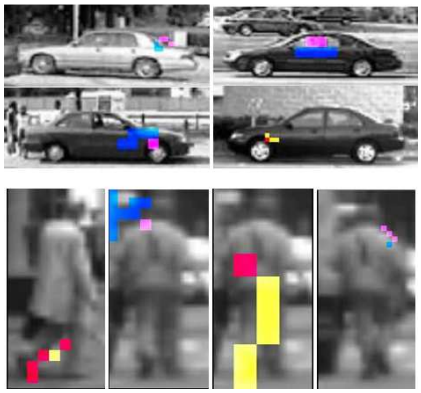


Figure 196 - Exemple de classificateur Connected-Control-Points

Dans LeViS sont actuellement implémentés ces 3 types ce classificateurs faibles, ainsi que l'algorithme Adaboost et sa variante cascadée.

Le lecteur désireux d'approfondir le sujet pourra se référer à (Meir, et al., 2003) et en ce qui concerne son application à la reconnaissance d'image à (Viola, et al., 2001) et (Abramson, 2005) ou encore (Schapire, 2009) qui contient une collection d'articles de référence mise à jour régulièrement.

c) Meta learner

Il ne s'agit pas vraiment d'un algorithme d'apprentissage en tant que tel, mais d'une fonctionnalité que LeViS implémente sous la forme d'un tel algorithme grâce à la grande généricité du code source de cet outil.

Il est parfois difficile de trouver les bons paramètres d'un algorithme (nombre de neurones de la couche cachée d'un réseau de neurones, nombre de points de contrôle à utiliser dans Adaboost,…) et il serait bon d'avoir un « outil » afin d'aider à les choisir.

Ainsi, au sein de LeViS a été implémenté un algorithme d'apprentissage qui, à l'instar de SamGT, teste successivement toutes les valeurs définies en paramètre d'un autre algorithme, et effectue un apprentissage pour chaque jeu de paramètres sur la même base de données. Les valeurs des paramètres testés sont sauvegardées dans un fichier, ainsi que les graphes d'apprentissage correspondants. Il suffit alors de consulter ces graphes en fin d'apprentissage pour choisir le meilleur jeu de paramètres à utiliser.

Le choix automatique du meilleur jeu de paramètres n'est pas aisé et soulève de nombreuses questions. En effet, il est possible, dans le choix de la topologie d'un perceptron multicouches par exemple, d'atteindre un meilleur taux d'erreur avec un grand nombre de neurones sur la couche cachée qu'avec un nombre moindre. Cependant si en utilisant moins de neurones, le taux d'erreur reste acceptable (ou est très peu supérieur au meilleur taux), la question de savoir laquelle des deux topologies est la meilleure pour l'application visée ne peut avoir de réponse automatique : meilleur taux d'erreur avec une forte charge calculatoire, ou taux d'erreur moins bon, mais algorithme plus rapide. Ce type de question peut se poser pour tous les algorithmes / jeux de paramètres à évaluer. C'est pourquoi, aucun processus de choix automatique n'a été implémenté au sein de ce Meta-Learner, surtout qu'il est rapide d'analyser visuellement un a un les graphes d'apprentissage.

5.3.4. Expérimentations

Dans un premier onglet (Figure 197), LeViS propose le chargement d'une base de données avec certaines options dont :

- La taille de ré-échantillonnage des exemples
- Les prétraitements à effectuer sur chacun des exemples (égalisation d'histogramme, conversion en image de gradient,…)

- Le canal sur lequel effectuer l'apprentissage (couleur, rouge, vert, bleu ou en niveau de gris)
- Le facteur de séparation pour créer les bases d'apprentissage et de test (typiquement 66%). La base de test servant à vérifier que l'algorithme arrive à correctement classer des exemples qui n'ont pas servit à l'apprentissage (i.e. des exemples « inconnus »).

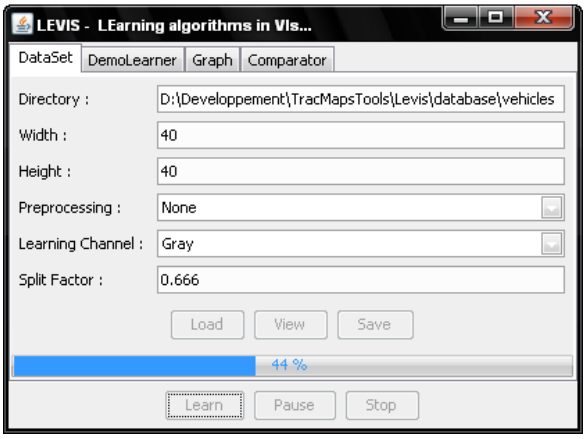


Figure 197 - Choix d'une base de données sous LeViS

Une contrainte majeure de nombreux algorithmes de classification d'images réside dans le fait qu'ils doivent traiter des données qui ont toujours les mêmes dimensions. Or, choisir une taille de ré-échantillonnage n'est pas aisé. Pour aider à ce choix, LeViS propose un outil visuel qui permet de visualiser dynamiquement l'image ré-échantillonnée par le dépacement de *sliders* (Figure 198).

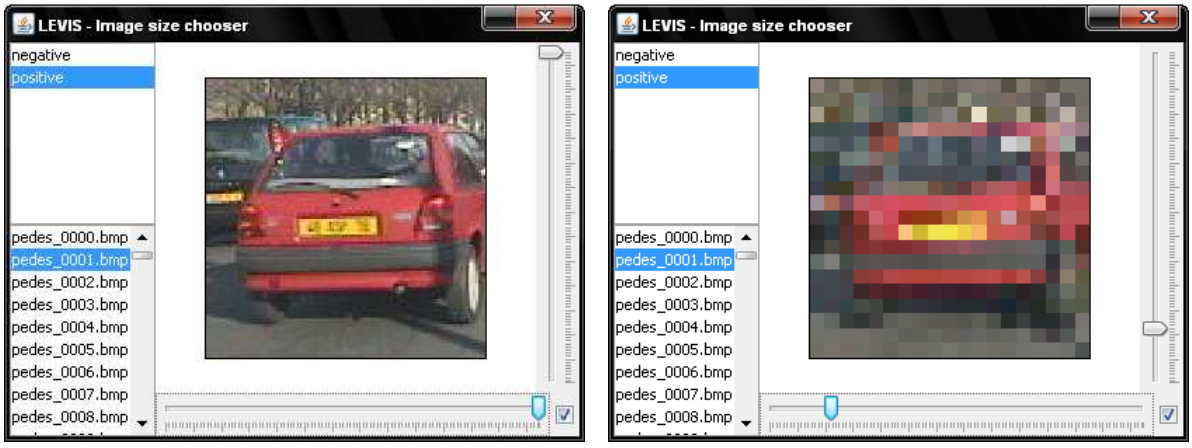


Figure 198 - Choix d'une taille de ré-échantillonnage sous LeViS : à gauche taille réelle de l'image, à droite image sous-échantillonnée

Voici à titre d'exemple les résultats obtenus avec les algorithmes d'apprentissage et de classification présentés plus haut dans les Exemple 5 et Exemple 6. Pour rappel ce classificateur calcule simplement la différence des valeurs de 2 pixels, et, si cette différence est supérieure à un seuil, classe l'exemple en tant que positif sinon en tant que négatif (ici en tant que voiture ou non voiture).

L'algorithme d'apprentissage trouve itérativement le meilleur classificateur en affectant des valeurs aléatoires aux positions des points et du seuil.

LeViS permet de visualiser en temps réel l'évolution du graphe d'apprentissage, i.e. les taux d'erreur de classification sur la base de test et d'apprentissage, courbes rouge et bleue, en fonction de l'itération (Figure 199). Une fois un classificateur appris, il est possible de visualiser, dans une nouvelle fenêtre, les exemples mal classés de la base de test ou d'apprentissage ainsi que, si il implémente des fonctions d'affichage, de visualiser le classificateur sur une image de la base (Figure 200 – Les deux points du classificateur sont affichés : en bleu, sous la voiture, et rouge au coin inférieur gauche de l'image).

Il est intéressant de voir que ces algorithmes simples arrivent à classifier une base de voiture avec un taux d'erreur de seulement 17%. Un tel taux d'erreur n'est pas admissible pour utiliser cet algorithme de classification directement dans un système ADAS, mais laisse envisager de pouvoir l'utiliser conjointement avec d'autres algorithmes (boosting,…).

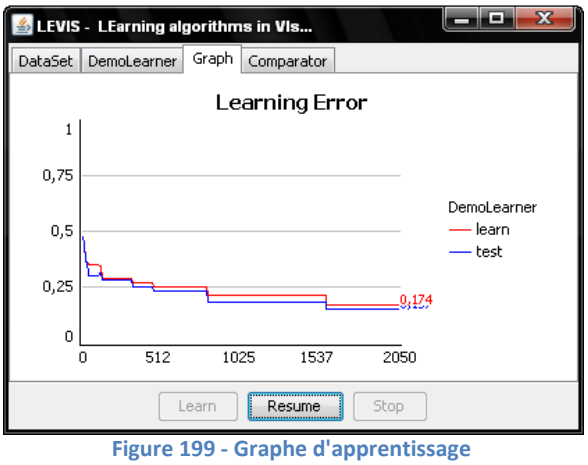


Figure 199 - Graphe d'apprentissage

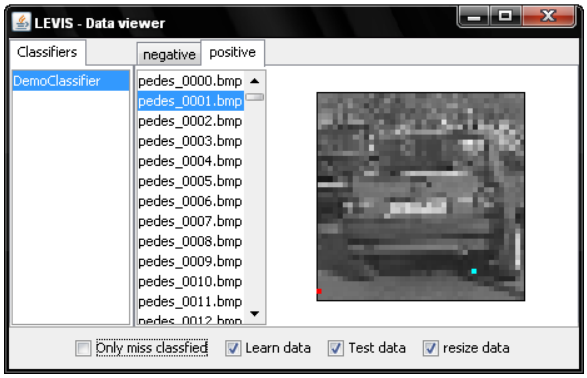


Figure 200 - Fenêtre d'affichage de la base de données et des classificateurs

En effet, l'utilisation de ce classificateur avec l'algorithme Adaboost permet, comme le montre la Figure 201, d'obtenir un taux d'erreur de classification de 0% sur la base d'apprentissage et de 3% sur la base de test. Le classificateur implémentant l'interface *Randomize* peut être utilisé avec l'algorithme d'apprentissage aléatoire multithreadé proposé dans LeViS. Le graphe d'apprentissage montre l'évolution de l'apprentissage de chaque thread de chacun des algorithmes d'apprentissage en cours d'exécution (i.e. ceux de l'algorithme d'apprentissage fort et ceux du faible). La Figure 202 montre l'ensemble des classificateurs faibles appris par Adaboost appliqués sur une image de la base.

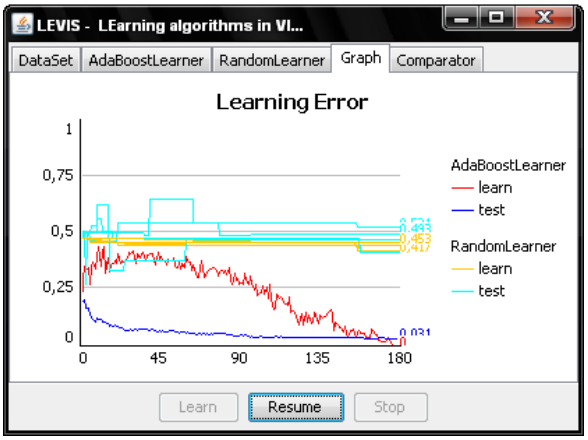


Figure 201 - Graphe d'apprentissage

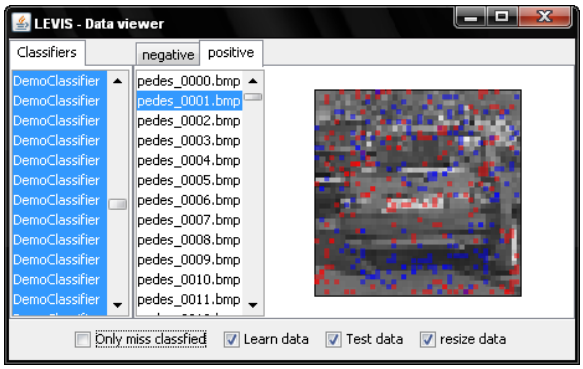


Figure 202 - Fenêtre d'affichage de la base de données et des classificateurs

Enfin, l'utilisation l'algorithme *MetaLearner*, permet par exemple d'aider au choix du nombre de neurones de la couche cachée d'un Perceptron Multicouches (Figure 203), qui est pour rappel, la principale variable « difficile » à trouver, l'algorithme d'apprentissage (la *Backpropagation*) se chargeant de trouver (apprendre) les poids des connexions entre les neurones. L'exemple est ici appliquée sur la base de chiffres ayant servi à l'apprentissage pour l'algorithme de détection visuelle des panneaux de limitation de vitesse présenté dans le premier chapitre de ce document. Le nombre d'itérations par apprentissage du réseau de neurones est limité à 750 pour cette étude.

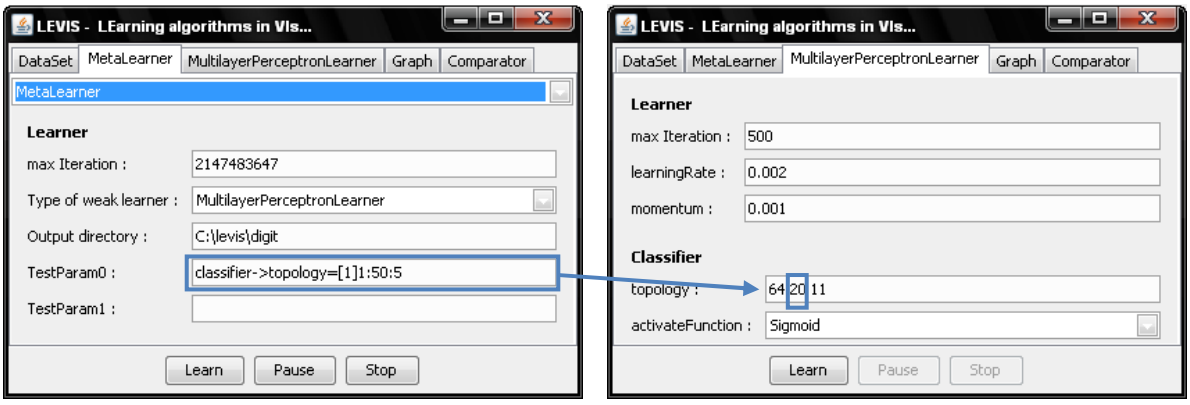


Figure 203 - Le *MetaLearner* sous LeViS, permet d'effectuer automatiquement des apprentissages avec des paramètres différents

Les résultats obtenus sont représentés par les Figure 204 à Figure 210. A partir de 20 neurones sur la couche cachée, la baisse du taux d'erreur sur les bases d'apprentissage et de test n'est plus significative (36% pour 20 neurones, 35% pour 30). C'est ce nombre de 20 neurones qui a été choisi pour le perceptron multicouche de reconnaissance des chiffres utilisé dans l'algorithme de détection des panneaux de limitation de vitesse.

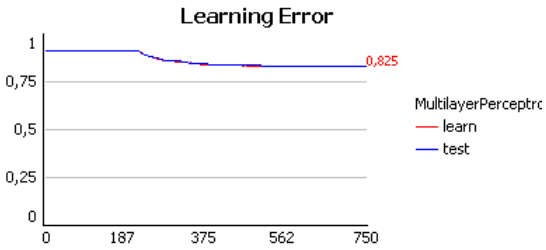


Figure 204 - Graphe d'apprentissage d'un Perceptron Multicouches à 1 neurone sur la couche cachée sur la base de chiffres

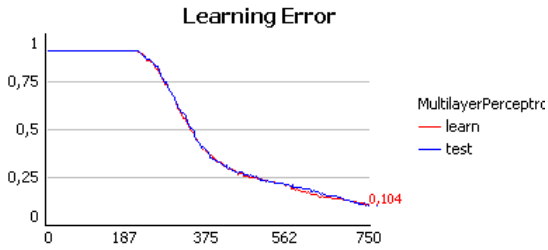


Figure 205 - Graphe d'apprentissage d'un Perceptron Multicouches à 11 neurones sur la couche cachée sur la base de chiffres

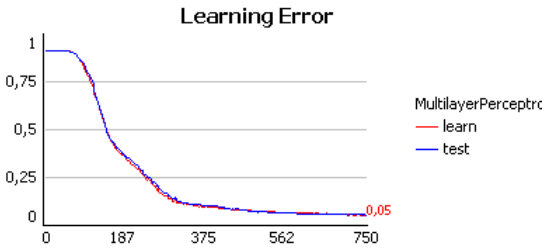


Figure 206 - Graphe d'apprentissage d'un Perceptron Multicouches à 16 neurones sur la couche cachée sur la base de chiffres

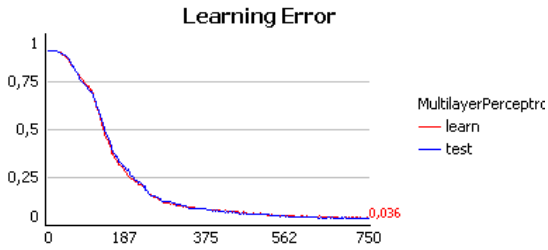


Figure 207 - Graphe d'apprentissage d'un Perceptron Multicouches à 21 neurones sur la couche cachée sur la base de chiffres

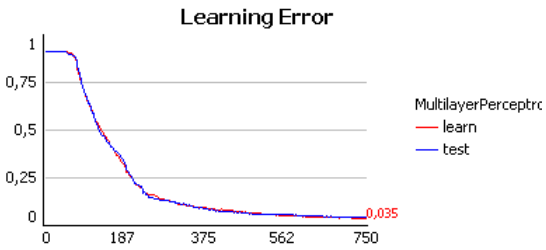


Figure 208 - Graphe d'apprentissage d'un Perceptron Multicouches à 31 neurones sur la couche cachée sur la base de chiffres

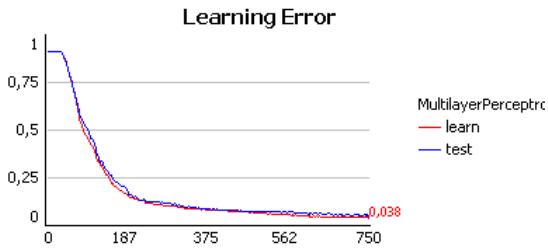


Figure 209 - Graphe d'apprentissage d'un Perceptron Multicouches à 41 neurones sur la couche cachée sur la base de chiffres

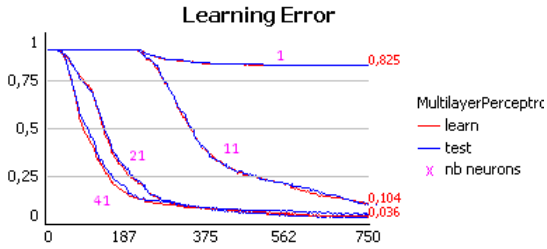


Figure 210 - Superposition des graphes d'apprentissage pour 1 à 41 neurones sur la couche cachée d'un Perceptron Multicouches sur la base de chiffres

5.4. Serveur d'acquisitions rtMaps

5.4.1. Présentation générale

Le logiciel rtMaps utilisé au laboratoire et pour mes travaux de recherche, est un logiciel permettant entre autres d'acquérir, d'enregistrer et de rejouer des bases de données multisensorielles. Ces bases de données peuvent donc contenir des séquences d'images, des fichiers son, des fichiers contenant des données issues de capteurs divers choisis par l'utilisateur et en fonction de l'application visée. Les séquences acquises au laboratoire sont essentiellement formées de données issues de capteurs couramment utilisés dans l'automobile, soit pour la détection de l'environnement soit pour relever les paramètres d'état du véhicule porteur. Il s'agit de : caméras, télémètres laser, radars, ultrasons, GPS, gyromètres, bus CAN, centrales inertielles, etc.

Tout cela explique que les bases de données peuvent présenter des contenus de tailles différentes et de « syntaxes » différentes et variées. Stocker et archiver tous ces enregistrements nécessite donc un espace disque important mais aussi un effort d'indexation et de référencement efficace et dédié.

Les utilisateurs de rtMaps, tout au moins au sein du laboratoire, n'ayant pas de lieu de partage structuré, se contentent généralement de sauvegarder leurs séquences personnelles (i.e. celles qu'ils ont acquises eux-mêmes pour leurs travaux de recherche) sur leurs propres ordinateurs. Ainsi, quand une autre personne a besoin d'une séquence particulière, il est amené, soit à demander à chacun si il n'aurait pas ce type de séquence, soit à aller ré-effectuer cet enregistrement, ce qui peut être, dans les deux cas, long et fastidieux.

Le Serveur d'Acquisition rtMaps (SAM) a ainsi été créé afin de répondre à ces besoins. Telle une bibliothèque ou plutôt une médiathèque, le serveur SAM présente un espace énorme dans lequel sont archivées et stockées des centaines de bases de données rtMaps. Celles-ci sont classées selon l'auteur de l'acquisition, la date, le contenu (c'est-à-dire les capteurs qui nourrissent chaque base) mais aussi le thème (c'est-à-dire le type d'environnement acquis, de quel temps, ...). Comme pour tout serveur d'archivage, l'utilisateur, après enregistrement, peut y rechercher des séquences selon divers critères et en ajouter de nouvelles, l'accès y étant facilité via l'utilisation d'un site intranet dédié et développé spécifiquement.

5.4.2. Rechercher une séquence

a) Par critères

Afin de faciliter la recherche d'une séquence parmi les centaines de gigaoctets de données déjà présents sur le serveur SAM, le site intranet propose une page de recherche permettant de retrouver une séquence de trois façons différentes (Figure 211).

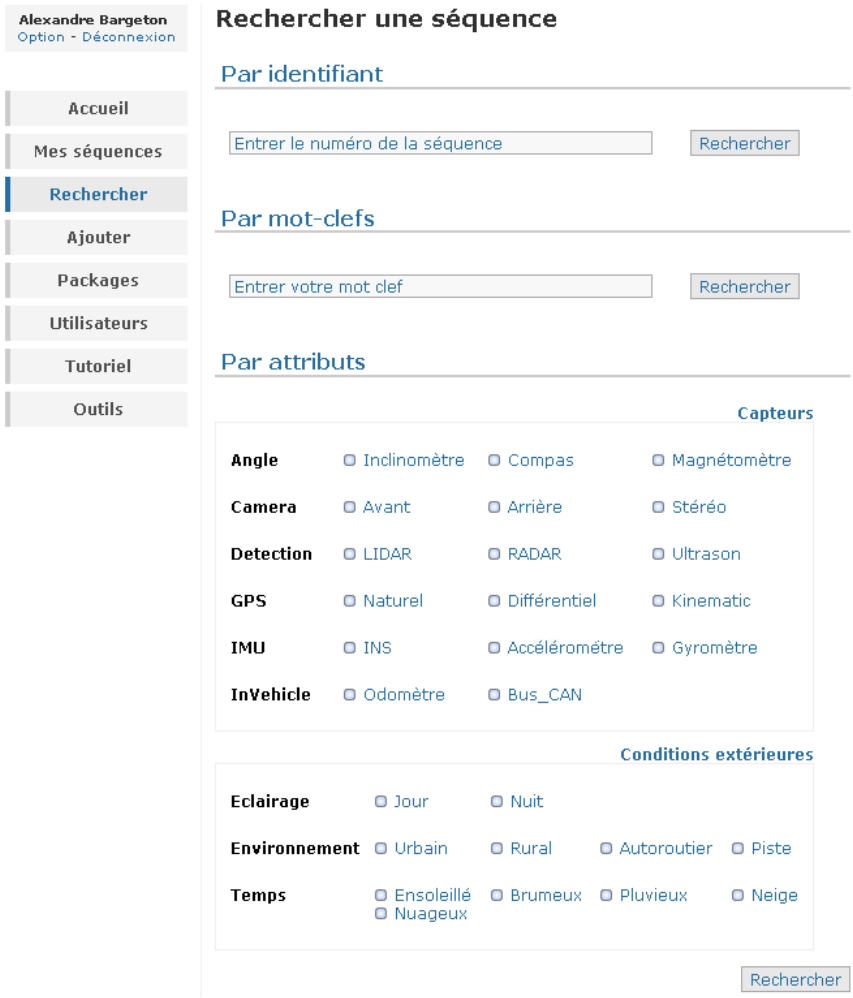


Figure 211 - Page de recherche de séquences du site intranet du server SAM

1. Soit l'utilisateur connaît l'identifiant de la séquence à rechercher, il pourra alors dans ce cas accéder directement à la séquence. Cette méthode simple, rapide et efficace, permet de partager des séquences facilement avec d'autres utilisateurs en diffusant directement leurs identifiants

2. Soit, si il a des critères précis sur les types de capteurs utilisés (caméra avant, laser,…) et de conditions extérieures lors de l'acquisition (de jour, avec de la pluie,…), l'utilisateur pourra alors effectuer sa recherche par attributs.

3. Soit il peut effectuer une recherche par mots clefs, qui seront retrouvés parmi les descriptions des séquences (par exemple : le nom d'un lieu, si on sait que l'acquisition y a été effectué) au cas où la liste des attributs disponibles ne suffirait pas à satisfaire sa requête.

b) Par liste

Une autre méthode pour retrouver facilement des séquences sur le serveur est l'utilisation des listes de séquences. Ainsi, sur le site intranet de SAM, il est possible de définir des listes de séquences à la manière des « favoris / bookmarks » des navigateurs web. Cela peut permettre, par exemple, de créer une liste de séquences par projet ou par type de travaux de recherche afin de les retrouver / partager au plus vite. Il est ainsi possible de regrouper toutes les séquences utilisées par exemple pour ses travaux de détections des panneaux de limitation de vitesse,…

Une liste peut bien entendu être créée, avec un nom spécifié par l'utilisateur, enrichie jour après jour de nouvelles séquences, et enfin une liste peut être supprimée (Figure 212).

Figure 212 - Gestion des listes de séquences et liste des séquences ajoutées sur le site intranet du server SAM

c) Par séquence ajoutée

Enfin, l'utilisateur a la possibilité de connaître toutes les séquences qu'il a lui même ajoutées. Généralement ce sont ces séquences auxquelles il accédera souvent ou auxquelles il fera référence afin de les partager avec ses collaborateurs, en leur fournissant directement leurs identifiants par exemple (Figure 212).

5.4.3. Télécharger une séquence

a) *Les droits d'accès*

Les séquences dont nous disposons au sein du laboratoire peuvent provenir de sources différentes :

- de nos propres enregistrements ;
- de nos partenaires (ex : Valeo, PSA LiViC, INRIA, …)

Il est toujours intéressant de pouvoir stocker le maximum de séquences provenant de toutes ces sources et de les proposer en une utilisation interne à tous les chercheurs du laboratoire. En effet, nos partenaires peuvent avoir des capteurs particuliers ou avoir enregistré des séquences sous des conditions spécifiques qu'il serait difficile de reproduire. Ainsi pouvoir proposer ces séquences aux autres membres du laboratoire peut permettre un gain de temps considérable pour les travaux de recherche du laboratoire.

Cependant, en gage de respect de la propriété intellectuelle, il faut pouvoir garantir que les données provenant d'un de nos partenaires industriels ne soient pas fournies à un autre.

Ainsi certaines séquences sont soumises à des droits d'accès où il sera demandé à l'utilisateur d'accepter les « accords de confidentialité » avant de pouvoir les télécharger et d'en respecter leurs conditions d'utilisation. Ces accords, où la provenance des séquences y est stipulée, sont bien entendu présentés à titre informatif, leur acceptation (via une case à cocher) permet de s'assurer au minimum que l'utilisateur en a bien eu connaissance (Figure 213).

Figure 213 - Droit d'accès et accord de confidentialité d'une séquence sur le site intranet du server SAM

b) *Les informations disponibles sur la séquence*

Une fois qu'une séquence a été retrouvée, et après acceptation des accords de confidentialité si il y a lieu, sont présentées sur une page distincte toutes les informations disponibles et utiles sur cette séquence (Figure 214) :

- Son identifiant (afin de pouvoir y accéder plus rapidement une prochaine fois, ou de l'ajouter dans une liste) ;
- Sa date d'ajout sur le serveur ;
- La personne qui a ajouté cette séquence, qui est potentiellement celle qui a effectué l'acquisition et qui pourra répondre aux éventuelles questions sur cette séquence (par exemple si elle dispose de fichiers de calibration non présents sur le serveur,…) ;
- Sa date et son heure d'acquisition (ce qui peut être utile pour en déduire des informations climatiques, par exemple un soleil rasant de crépuscule) ;
- La taille physique de la séquence (en octet), qui peut être utile avant le téléchargement, les séquences pouvant atteindre quelques giga-octets ;
- sa durée (utile pour pouvoir discriminer, si l'on recherche une séquence de plusieurs dizaines de minutes et non de quelques secondes par exemple)
- Sa description, renseignée par l'auteur de cette séquence, et proposant des informations complémentaires aux attributs.

De plus, si une séquence possède une ou plusieurs acquisitions vidéo, celles-ci pourront être rejouées en ligne (directement depuis le site), en une taille d'image réduite, permettant de vérifier facilement si cette séquence est bien celle recherchée (Figure 214).

Figure 214 - Les informations relatives à une séquence sur sa page dédiée du site intranet du server SAM

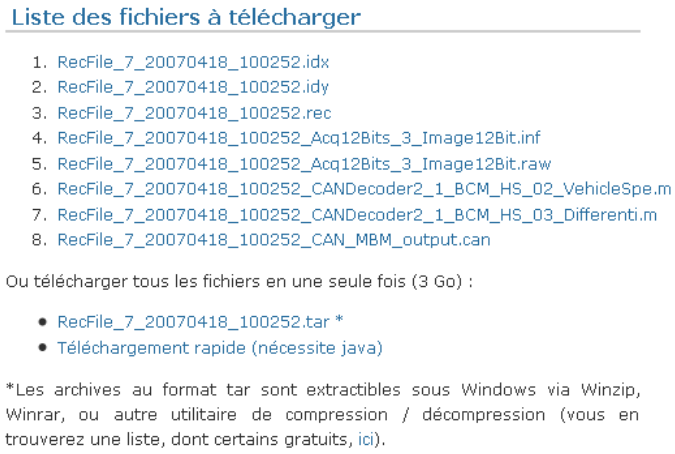
c) *Le téléchargement*

Afin de récupérer les fichiers d'une séquence, trois modes de téléchargement sont proposés (Figure 215) :

1. Fichier par fichier (transfert HTTP): les fichiers constituant la séquence sont listés et il est possible de ne télécharger seulement que celui voulu.

2. Sous forme d'archive au format tar (transfert HTTP) : pour télécharger tous les fichiers en une seule fois. Un fichier *tar* n'est pas compressé, c'est simplement une concaténation de tous les fichiers en un seul. La taille de cette archive pouvant donc être assez conséquente, celle-ci est rappelée afin de vérifier l'espace disque disponible avant le téléchargement.

3. Via une page de téléchargement dédiée (transfert FTP) : les deux précédents modes de téléchargement utilisent le transfert via le protocole HTTP. Celui-ci, pour des raisons techniques pour le transfert de volumes importants, est plus lent que le protocole FTP. En effet, en HTTP, les données sont encapsulées et la réception des paquets est vérifiées, alors qu'en FTP, seul l'envoi des données est effectué.


Liste des fichiers à télécharger

1. RecFile_7_20070418_100252.idx
2. RecFile_7_20070418_100252.idy
3. RecFile_7_20070418_100252.rec
4. RecFile_7_20070418_100252_Acq12Bits_3_Image12Bit.inf
5. RecFile_7_20070418_100252_Acq12Bits_3_Image12Bit.raw
6. RecFile_7_20070418_100252_CANDecoder2_1_BCM_HS_02_VehicleSpe.m
7. RecFile_7_20070418_100252_CANDecoder2_1_BCM_HS_03_Differenti.m
8. RecFile_7_20070418_100252_CAN_MBM_output.can

Ou télécharger tous les fichiers en une seule fois (3 Go) :

- RecFile_7_20070418_100252.tar *
- Téléchargement rapide (nécessite java)

*Les archives au format tar sont extractibles sous Windows via Winzip, Winrar, ou autre utilitaire de compression / décompression (vous en trouverez une liste, dont certains gratuits, ici).

Figure 215 - Téléchargement des fichiers d'une séquence sur le site intranet du server SAM

5.4.4. Ajouter une séquence

a) Etiqueter et décrire une séquence

Beaucoup d'informations peuvent être automatiquement déduites des fichiers qui seront ajoutés sur le serveur (taille, durée,…), mais de nombreuses autres doivent être correctement renseignées par l'utilisateur. Ainsi, cette étape est la plus critique pour le bon fonctionnent du serveur SAM (i.e. pouvoir rechercher une séquence efficacement sur le serveur).

A l'instar du formulaire de recherche, le formulaire de description d'une séquence propose exactement les mêmes zones à renseigner : attributs, conditions climatiques, et description.

b) Spécifier le niveau de confidentialité

Pour garantir un accès plus ou moins restreint à une séquence, il est nécessaire de lui définir son niveau de confidentialité lors de son ajout sur le serveur, qui pourra être :

- Aucune
  La séquence pourra être partagée avec tout le monde (y compris avec nos divers partenaires industriels)

- Utilisation interne seulement
  La séquence ne pourra être utilisée que pour une recherche interne au laboratoire et avec le partenaire industriel spécifié.

c) *Ajout des fichiers*

L'ajout des fichiers sur le serveur ne se fait ici que via le protocole FTP (le protocole HTTP ne supportant pas l'envoi de fichier volumineux car n'étant pas conçu pour). Une page dédiée contenant son logiciel de transfert (un applet Java), s'occupe de façon transparente pour l'utilisateur de la connexion et du transfert des fichiers vers le serveur SAM. Là aussi l'utilisation du protocole FTP permet d'atteindre des vitesses de transfert assez rapide (Figure 216 – transfert à 9Mo/s).

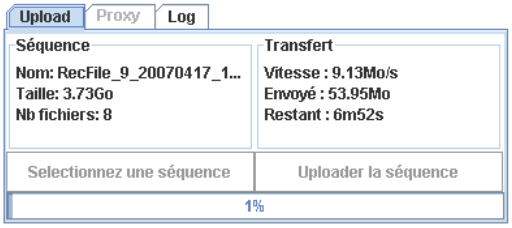


Figure 216 - Transfert d'une séquence sur le serveur SAM

Une fois le transfert terminé, le serveur enregistre toutes les informations fournies par l'utilisateur (description, attributs,…) et affiche sur une nouvelle page l'identifiant unique de cette séquence.

5.4.5. D'un point de vue plus technique

Le serveur (i.e. la plateforme logicielle) et le site intranet ont été conçus sur mesure pour les besoins de SAM et pour l'intégration dans le réseau informatique existant du laboratoire.

Il existait déjà un serveur web (i.e. une machine avec un logiciel serveur HTTP) pour l'hébergement du site internet du laboratoire. Cependant ce serveur n'a pas été conçu pour être un serveur de stockage massif de données. Ainsi une nouvelle machine a été acquise, comportant 4 disques durs de 1To chacun, montés en RAID (technologie qui permet entre autre une plus grande tolérance aux pannes via le stockage redondant des données sur plusieurs disques).

Comme le résume la Figure 217, le serveur SAM est donc en fait constitué de deux ordinateurs distincts : l'ancien serveur web où est hébergé le site intranet et le nouveau serveur de stockage où sont enregistrées les séquences ^{rt}Maps. Le dialogue (i.e. le transfert des données) entre ces deux machines se faisant via des communications dédiées décrites dans les paragraphes suivants et n'est diffusé qu'à un utilisateur authentifié sur le site intranet (login + mot de passe) que lors de l'ajout ou du téléchargement d'une séquence par celui-ci. Un quelconque vol du mot de passe FTP n'est alors que peu utile car celui-ci ne sera plus valide quelques secondes après.

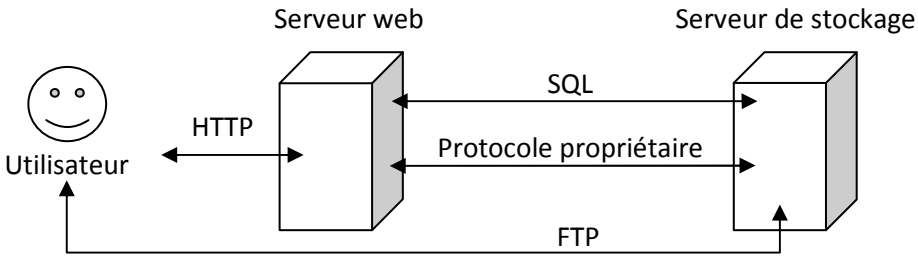


Figure 217 - Fonctionnant du serveur SAM

Tout d'abord, sur le serveur de stockage est mis en place un serveur de transfert de fichier FTP (logiciel externe développé par des tiers) pour assurer le transfert des séquences (ajout et téléchargement) depuis ou vers le serveur. Afin de garantir un niveau de sécurité optimal, le mot de passe du compte FTP est changé automatiquement et aléatoirement toutes les minutes.

Une base de données MySQL avec serveur d'accès pour y effectuer des requêtes est aussi mise en place sur le serveur de stockage. Cette base de données sert à stocker les attributs des séquences renseignées par les utilisateurs, ainsi que les informations des utilisateurs (nom, prénom, login, mot de passe crypté, adresse mail,…).

Ensuite la communication entre le serveur web (plus spécifiquement avec le site web) et le serveur de stockage est établie via un protocole propriétaire spécifiquement mis en place. Un serveur (logiciel) dédié a donc été développé (en C). Ce protocole / serveur permet d'effectuer des recherches physiques sur le serveur de stockage comme des recherches sur les noms de fichiers, sur les tailles,… mais permet aussi d'extraire et de transférer les images (après réduction de taille) des acquisitions vidéo des séquences qui pourront ensuite être visualisées sur le site web sous forme de vidéo ; et de créer à la volée, les archives au format *tar*. En effet, ces archives ne sont jamais stockées physiquement sur le serveur (cela doublerait l'espace disque nécessaire pour l'archivage des séquences) mais ne sont générées et transférées sur le réseau qu'à la demande d'un utilisateur.

Le site intranet a lui aussi été développé sur mesure pour répondre au plus près aux besoins requis par SAM. Ainsi, une première couche logicielle en PHP permet de communiquer avec le serveur de stockage soit via des requêtes SQL, soit via le protocole spécifiquement mis en place et de générer dynamiquement les pages web du site intranet. Cela permet par exemple de gérer une connexion sécurisée par mot de passe des utilisateurs avant qu'il puisse avoir accès aux données et avant de pouvoir transférer des séquences.

5.5. Perspectives

L'ensemble des outils présentés ici est maintenant couramment utilisé au sein du laboratoire, gage d'une réelle utilité pour les travaux de recherche de chacun. Certains sont aussi utilisés par nos partenaires industriels (Valeo,…) et au sein de projets collaboratifs nationaux (LoVE,…) regroupant d'autres laboratoire de recherche ainsi que d'autres industriels du milieu automobile.

Ces outils sont proposés, en interne au laboratoire, sur un serveur de gestion de version des codes source (serveur Subversion - SVN). Ceci permet aux autres doctorants ou chercheurs du laboratoire, de pouvoir les récupérer et les utiliser directement ; mais aussi de pouvoir les modifier et les améliorer ; puis de mettre à jour l'ensemble du code source sur le serveur. Les modifications seront alors « automatiquement » répercutées auprès des autres membres du laboratoire qui bénéficieront des améliorations apportés par chacun.

Plusieurs doctorants du laboratoire utilisent activement ces logiciels, et notamment ceux du groupe ADAS, qui y ont déjà apporté quelques améliorations (code Adaboost pour LeViS, génération d'un fichier de statistiques plus élaborés dans SamGT,…), gage d'un réel engouement pour ces outils et de leur utilité.

Enfin, la mise en commun des codes sources Java des logiciels LeViS et SamGT permet d'utiliser les même codes élémentaires (lectures des vidéo des séquences ^{rt}Maps par exemple) pour chacun des logiciels, sans les dupliquer ; mais aussi a permis de réaliser des fonctionnalités plus avancées, comme par exemple l'extraction automatique des parties d'image représentatives de classes négatives (par comparaison avec une fichier de vérité terrain de SamGT) pour la création de bases d'apprentissage dynamiques dans LeViS utile pour certains algorithmes (Adaboost Cascadé,…).