Académie de Montpellier
Université Montpellier II
Sciences et Techniques du Languedoc

MÉMOIRE BIBLIOGRAPHIQUE DE

MASTER M2

effectué au Laboratoire d'Informatique de Robotique
et de Micro-électronique de Montpellier

Spécialité : **IPS**

La stéganalyse à grande échelle

par **Jérôme Pasquet**

Sous la co-direction de **Sandra Bringay et Marc Chaumont**

Table des matières

1 Introduction

La stéganographie est l'art de dissimuler de façon anodine un message dans un document. Par opposition, la stéganalyse est l'art de déceler la présence de ce message. Les technologies utilisées pour la stéganographie se sont considérablement améliorées ces dernières années. Le plus souvent, le stéganalyste se place dans un cadre favorable pour décider si une image est *stego*[1] ou *cover*[2], pour cela il pose des hypothèses fortes décrivant le scénario. Le plus utilisé est celui de *clairvoyance* [15]. Dans ce scénario, le stéganalyste connaît l'algorithme d'insertion utilisé et le type des images (appareil photo, lieux, etc). Dans le cas d'une analyse automatique de trafic, c'est-à-dire où deux individus échangent librement des documents, cette hypothèse n'est pas vraiment réaliste. Le stéganalyste ne peut pas avoir une connaissance parfaite du type d'image utilisée par le stéganographe. En effet, le stégnographe ne va pas insérer des messages dans des supports ayant les mêmes caractéristiques, comme des photographies issues du même appareil photo. Il y aura donc différence entre les types d'images que le stéganalyste connaît et celles que le stéganographe utilise pour l'insertion. Ce problème s'appelle le *cover-source mismatch*. Nous le détaillerons davantage par la suite. C'est dans ce contexte difficile mais plus réaliste que se situe le stage.

Pour définir si une image contient un message, nous allons dans notre étude utiliser des méthodes de *machines learning*. La plus utilisée est la méthode des Séparateurs à Vaste Marge (SVM) [18, 1, 12]. Or, les algorithmes de stéganographie s'améliorant toujours plus vite, la tendance est d'utiliser des vecteurs caractéristiques plus grands [7]. De plus, pour traiter le cas avec *cover-source mismatch*, une hypothèse proposée par Ivans Lubenko et Andrew D. Ker est d'effectuer l'apprentissage sur de plus grands volumes [14]. La complexité des SVM étant quadratique, avec ces contraintes, elle devient trop coûteuse. Dans le cadre de ce stage et afin de ne pas rencontrer les mêmes difficultés, nous allons expérimenter les classifieurs à complexité linéaire tel que : l'ensemble average perceptron [13] et l'ensemble classifieur de Kodovský [10].

Nous présenterons dans une première partie le mécanisme global de stéganographie et de stéganalyse. Dans cette même partie, nous expliquerons brièvement comment fonctionne l'insertion. Pour cela, nous détaillerons un algorithme d'insertion dans le domaine spatial nommé HUGO [17]. Dans une seconde partie, nous présenterons d'une façon très générale les différentes méthodes de stéganalyse. Nous verrons tout particulièrement les méthodes basées sur des algorithmes d'apprentissage. Pour finir, nous définirons les

1. Une image stego est une image contenant un message
2. Une image couverture est une image ne contenant pas de message

objectifs du stage, ainsi que les perceptives envisagées pour répondre aux problèmes du *cover-source mismatch*.

2 Concepts généraux

Dans cette section, nous présenterons tout d'abord les principes fondamentaux et le fonctionnement de la stéganographie. Puis, nous formaliserons le problème de la stéganalyse. Nous finirons par une brève présentation d'un système d'insertion.

2.1 Le scénario historique de la stéganographie et de la stéganalyse

La stéganographie et stéganalyse est un jeu faisant intervenir trois participants. Deux stéganographes, s'appelant historiquement Alice et Bob, veulent établir une communication secrète. Le stéganaliste, appelé Eve, analyse les échanges entre Alice et Bob et peut stopper ces échanges s'il y a présence d'un message caché [4].

Par exemple, Alice peut envoyer un ensemble d'images à Bob par e-mail. Toutes les images envoyées seront visionnées et analysées par Eve. Si Eve détecte la présence d'un message, elle peut interdire la communication entre Alice et Bob. Par conséquent, Alice et Bob doivent cacher le message dans l'image au niveau visuel, sémantique et statistique.

Dans notre étude, nous nous plaçons dans le scénario où Eve connaît uniquement l'algorithme d'insertion utilisé ainsi que la quantité de bit échangé. Eve est confrontée au problème du *cover-source mismatch* c'est-à-dire elle ne connaît ni le type, ni la sémantique des images utilisées par Alice et Bob. Ce scénario est proche d'un cas réaliste d'analyse de trafic entre Alice et Bob. Nous jouerons le rôle de Eve et proposerons des algorithmes et solutions permettant de lutter contre le problème du *cover-source mismatch*.

2.2 Modélisation du problème de stéganalyse pour détecter les messages

Pour cette étude, nous allons utiliser les notations classiques suivantes : soit $\mathbf{x} = (x_1, x_2, .., x_n)$ une image composée de n pixels avec x_i ($i \in \{1..n\}$) le pixel d'index i dont la valeur appartient à l'intervalle $\{0..255\}$. Soit $\mathbf{x}$ une image *cover* et $\mathbf{y}$ la même image *stego*. Les dimensions des images sont toutes identiques, de largeur n_1 et de hauteur n_2 avec $n_1.n_2 = n$. Le vecteur caractéristique d'une image sera noté $\mathbf{f} = (f_1, f_2, .., f_d) \in \mathbb{R}^d$, avec d le nombre de caractéristiques. Nous notons $\mathbf{f}_{red} \in \mathbb{R}^{d_{red}}$ le vecteur issu du sous-échantillonage de $\mathbf{f}$ de dimension d_{red}. Nous définissons également un

label $\mathscr{L} = \{-1, 1\}$ valant 1 si l'image est *cover* et -1 si l'image est *stego*. N représente le nombre total d'images. La stéganalyse consiste à partir du vecteur caractéristique $\mathbf{f}$ d'une image, à lui associer un label appartenant à $\mathscr{L}$.

2.3 Stéganographie

Le but du stéganographe est de transmettre un message de façon indétectable. En fonction du type de l'image certains algorithmes sont plus adaptés. Par exemple, dans le cas d'une image jpeg, les algorithmes insérant dans le domaine fréquentiel comme nsF5 [8] sont apropriés. Dans notre étude, les images ne seront pas compressées, nous utilisons un algorithme d'insertion spatial. La première méthode adaptative, c'est-à-dire prenant en considération la détectabilité de chaque pixel, a été proposée via l'algorithme HUGO (Highly Undectable Stego) [17]. Celui-ci fut présenté lors de la compétition BOSS [3] en 2010.

On peut shématiser le fonctionnement de HUGO en deux grandes étapes : 1- création d'une carte de détectabilité, c'est-à-dire attribuer à chaque pixel la probabilité inverse de détection suite à une modification. 2- Insertion en modifiant certains pixels de la carte de détectabilité tout en minimisant la distorsion totale, c'est-à-dire en minimisant la détection des altérations.
La détermination des distorsions se fait en calculant les caractéristiques (voir annexe B) issues d'un filtre passe haut de SPAM [16]. Au total, lors de l'insertion via HUGO le calcul de minimisation s'effectue sur 10^7 caractéristiques. C'est pourquoi l'insertion sera particulièrement difficile à détecter [17] et il sera nécessaire de compléter les caractéristiques de SPAM avec *Rich Models* [7].

Lors du stage, nous tenterons de déterminer si une image est *cover* ou *stego* suite à une insertion via l'algorithme HUGO. Dans la section suivante, nous présenterons différentes façons de détecter une insertion en nous intéressant particulièrement aux méthodes par apprentissage.

3 Stéganalyse : l'art de détecter les messages

Dans cette section, nous allons en premier présenter différentes familles de stéganalyses que nous illustrerons par un exemple. Ensuite, nous nous focaliserons sur les méthodes par apprentissage automatique faisant un état de l'art de celles-ci.

3. Break Our Steganography System : http ://exile.felk.cvut.cz/boss/BOSSFinal/

3.1 Grandes familles des méthodes de stéganalyse

Les méthodes de stéganalyse peuvent répondre à deux objectifs : déterminer si une image contient un message ou déterminer la taille de ce message. Il existe deux grandes familles. 1- Les méthodes spécifiques ou structurelles sont celles qui ciblent un algorithme donné d'insertion. Elles repèrent un ensemble de propriétés statistiques qui varie d'une image *cover* à une image *stego*. 2- Les méthodes génériques utilisent des données d'entrainement afin de créer par apprentissage un modèle de classification.

Pour donner un exemple d'approche structurelle, nous nous interressons à une méthode de stéganalyse particulière aux images comportant des palettes [4] (type .gif) qui consiste à étudier la distribution des paires de pixels [5]. L'insertion de type LSB [5], dans ce cas-là, correspond à un changement d'index [6]. Sur une image classique, la quantité v_{2i} qui comptabilise la moyenne des pixels d'index 2i (c_{2i}) et 2i+1 (c_{2i+1}) devrait être imprédictible. L'idée est donc d'utiliser le χ^2 entre cette quantité v_{2i} et c_{2i} :

$$\chi^2 = \sum_{i=0}^{nb\ paires} \frac{(c_{2i} - v_{2i})^2}{v_{2i}} \tag{1}$$

L'inversion a comme conséquence globale de rendre proche la probabilité des pixels $2i$ avec celle des pixels $2i+1$ [4]. Par conséquent, plus χ^2 est faible, plus la probabilité que l'image soit stégo est importante. Les conséquences de ce type d'insertion sont représentées dans la figure 1.

HUGO, de part son fonctionnement (voir section 2.3), ne provoque pas de changements statistiques facilement détectables sur l'image [17]. Il est nécessaire d'utiliser une autre approche par apprentissage beaucoup plus générique. Les méthodes que nous décrirons par la suite sont de ce type.

3.2 Les méthodes d'apprentissage automatique (Machine learning)

Dans cette sous-section, nous détaillons les différentes méthodes d'apprentissage automatique, appliquer à la stéganalyse. Lors du stage, nous implémenterons et proposerons des améliorations de ces méthodes par apprentissage.

4. Une palette correspond à une liste d'index, chacun faisant référence à une couleur.
5. Une insertion de type LSB (least significant bit) correspond à un changement du bit de poids faible, et donc le pixel varie de un dans un sens donné.
6. Un index est une référence vers une couleur de la palette.

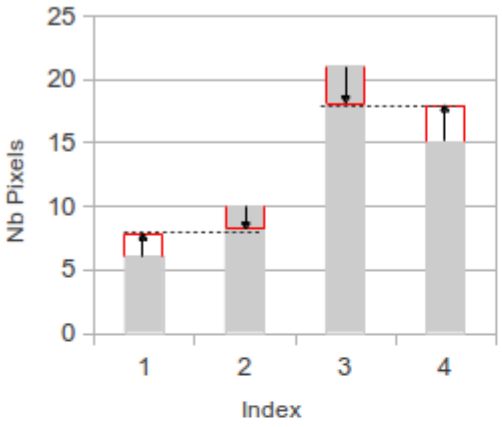

FIGURE 1 – L'histogramme suivant représente la répartition des pixels pour quatre valeurs d'index. Les flèches indiquent comment évolue le modèle après l'insertion.

3.2.1 SVM

Les Machines à Vecteurs de Supports (SVM) [1, 12] permettent d'obtenir une frontière [7] (optimale) pour séparer les données en deux classes *stego* et *cover*. Toutefois, certaines données ne sont pas linéairement séparables. Dans ce cas-là, il est nécessaire d'utiliser un espace de redescription (c'est-à-dire un espace de dimension supérieure). La frontière (ou hyperplan) sera ensuite recherchée dans ce nouvel espace de dimension supérieure. Pour se faire, if faut utiliser le *kernel trick*, c'est-à-dire une fonction noyau permettant de généraliser l'approche linéaire [1]. La classification par SVM fonctionne très bien dans le cas binaire, classification *cover* ou *stego*. Toutefois bien que présentant de bons résultats dans le cas des petits jeux de données, son utilisation est impossible pour de grands jeux de données. Pour nos expérimentations, nous utiliserons un noyau non linéaire décrit par Ivans Lubenko et Andrew D. Ker de type gaussien[12]. La complexité du noyau gaussien est quadratique. Avec l'utilisation de libsvm [8] la complexité varie de $O(d.N^2)$ à $O(d.N^3)$ en fonction de l'utilisation de la mémoire cache, avec d la dimension des vecteurs caractéristiques et N le nombre de données d'entrainement.

7. La frontière représente dans un espace à X dimensions la séparation entre les différentes classes.

8. http ://scikit-learn.org/dev/modules/svm.html

3.2.2 Average Perceptron

Pour gérer les grands jeux d'apprentissage, une solution consiste à utiliser un classifieur linéaire. Comme le montrent Ivans Lubenko et Andrew D. Ker [13] en 2012, l'approche par perceptron est très prometteuse. Un perceptron ou neurone, est un classifieur *online*[9] permettant de séparer linéairement les données [3]. Le neurone est consistué d'un vecteur poids, c'est-à-dire d'un vecteur $\mathbf{w} \in \mathbb{R}^d$. Ce vecteur représente la séparation entre les images *cover* et *stego*, initialement ce vecteur vaut $\vec{0}$ (vecteur nul).

L'apprentissage s'effectue par mises à jour successives. La mise à jour comprend le vecteur caractéristique $\mathbf{f}_i \in \mathbb{R}^d$ avec $i \in \{0..N\}$, N étant le nombre d'images dans la base de données et le type de l'image $l_i \in \mathscr{L}$. L'algorithme classe le vecteur $\mathbf{f}_i$, le résultat est noté z. Pour cela, on analyse la position du point $\mathbf{f}_i$ par rapport à la séparation $\mathbf{w}$, voir formule 2 :

$$z = sign(\mathbf{w}.\mathbf{f}_i) \tag{2}$$

Avec *sign* la fonction retournant le signe.

Une représentation 2D de cette classification est présentée dans la figure 2. En effectuant la projection de la formule 2, nous avons représenté les deux possibilités. Sur la figure de gauche, on voit que la projection de $\mathbf{f}$ est positive sur $\mathbf{w}$, comme le label est 1 il n'y aura aucune mise à jour. La figure de droite représente un deuxième scénario où la projection est négative sur $\mathbf{w}$, par conséquent il y aura une mise à jour.

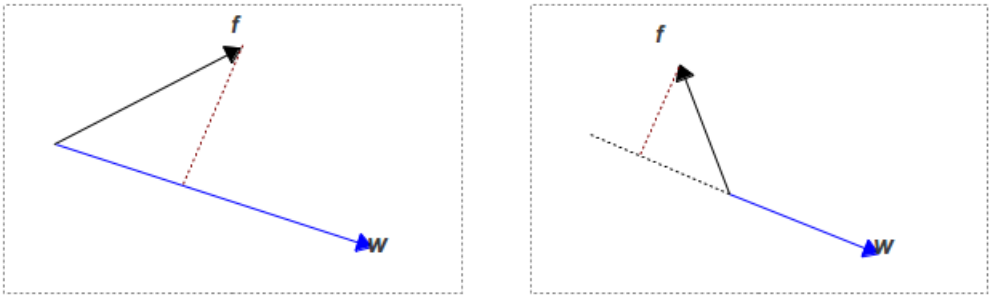

FIGURE 2 – Soient $\mathbf{w}$ le vecteur poids du perceptron et $\mathbf{f}$ le vecteur caractéristique à classer possédant un label $\mathscr{L}$ de 1.

Deux cas sont possibles : 1- l'algorithme de classification a correctement placé $\mathbf{f}_i$: aucune mise à jour n'est effectuée. 2- Sinon il faut modifier le poids

9. Un algorithme online ou en ligne est un algorithme qui reçoit ses données d'apprentissages pré-découpées. Il les assimile itérativement.

du neurone, comme l'indique la formule 3.

$$\begin{cases} si\ l_i = z & \text{alors aucune mise à jour} \\ si\ l_i \neq z & \text{alors } \mathbf{w} = \mathbf{w} + l_i.\mathbf{f}_i \end{cases} \tag{3}$$

L'évolution d'un perceptron est schématisée en deux dimensions dans la figure 3 : la figure 3.B représente l'évolution de la figure 3.A après la mise à jour de v_1. La figure 3.C représente la mise à jour v_2 réalisée à partir de la figure 3.B. Notons que sur la figure 3.B v_3 n'apportera aucune mise à jour car son label serait prédit.

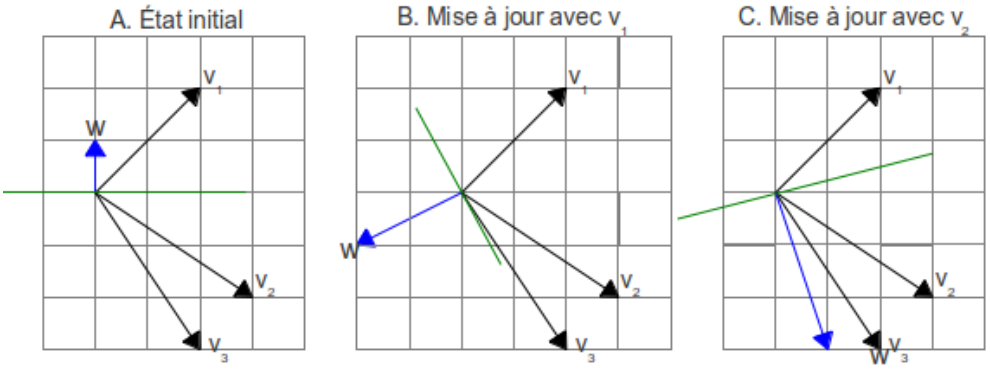

FIGURE 3 – Dans cette illustration, $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ sont des vecteurs quelconques avec respectivement -1, 1 et 1 comme classe $\mathscr{L}$. $\mathbf{w}$ est le poids du neurone étudié, la droite en vert représente la droite orthogonale à $\mathbf{w}$.

Comme l'a prouvé Novikoff [3], tout perceptron converge après un nombre fini d'itérations si les données sont linéairement séparables. Une autre version du perceptron existe : le *batch iterated perceptron* [13], celui-ci consiste à répéter k fois les j premières mises à jour, k et j étant des paramètres fixés par l'utilisateur. Ce processus permet d'accelérer la convergence. La complexité de chaque mise à jour est $O(d)$, ce qui équivaut pour l'apprentissage total à une complexité de $O(d.N)$.

3.2.3 Ensemble classifieur

Une autre approche, proposée par Kodovsky en 2011 consiste à utiliser un ensemble de classifieur [10, 9]. Le fonctionnement est simple et consiste à obtenir un vote de m classifieurs de faible complexité. Dans cet article le vote est de type majoritaire. Notons qu'il est possible de pondérer le vote de chaque classifieur [2].

L'ensemble classifieur s'apparente donc à une forêt d'arbres décisionnels, c'est-à-dire un classifieur qui exécute de multiples sous-classifieurs sur des données légèrement différentes. Deux paramètres sont à définir : le nombre L représentant le nombre de classifieurs faibles et la taille d_{red} de chaque sous-échantillonage du vecteur caractéristique $\mathbf{f}$ (utilisé en entrée de l'ensemble classifieur). Nous définissons, chaque classifieur faible comme étant une fonction h_i avec $i \in \{0..L\}$, telle que :

$$\begin{array}{rccc} h_i(\mathbf{f}): & \mathbf{f}_{red} & \rightarrow & \mathscr{L} \\ & \mathbb{R}^{d_{red}} & \rightarrow & \{-1, 1\} \end{array} \tag{4}$$

La décision finale $S \in \mathscr{L}$ par vote majoritaire de l'ensemble classifieur, pour un vecteur $\mathbf{f}$, est définie dans l'équation 5 :

$$S = sign(\sum_{i=0}^{L} (h_i(\mathbf{f}))) \tag{5}$$

Avec *sign* la fonction retournant le signe.

Chaque classifieur de base voit son apprentisssage s'effectuer sur un unique sous-échantillon de $\mathbf{f}$, l'apprentissage se fait indépendamment des autres classifieurs. La figure 5 résume le processus global d'apprentissage de l'ensemble classifieur. Notons que ces classifieurs faibles peuvent être un neurone ou n'importe quel autre type de classifieur.

Dans leur article Kodovský et Fridrich proposent une méthode d'automatisation du choix de L et d_{red}. Pour cela, ils utilisent une technique *bootstrap*, qui consiste à apprendre sur un certain nombre d'échantillons $N_{simulations}$ de la base de données. Pour la suite 67% de $N_{simulations}$, noté $N^a_{simulations}$ est utilisé pour l'apprentissage et 33%, noté $N^b_{simulations}$ est utilisé pour les tests. Dans un premier temps, ils définissent l'erreur dite *out-of-bag*, noté E_{OOB} qui représente pour chaque couple d'images $\mathbf{x}_i$ et $\mathbf{y}_i$ avec $i \in \{0..N^b_{simulations}\}$ la moyenne de la somme des erreurs de chaque classifieur faible. Notons que cette formule est ici modifiée par rapport à l'article afin d'homogénéiser les

notations.

$$E_{OOB} = \frac{1}{2.N^b_{simulations}} \sum_{i=0}^{N^b_{simulations}} \left(1 - \frac{h(\mathbf{x}_i) - h(\mathbf{y}_i)}{2}\right) \tag{6}$$

La première étape consiste à déterminer le nombre L de classifieur faible. Dans l'article [10], les auteurs constatent que pour un nombre fixé d_{red}, l'erreur *out-of-bag* converge asymptotiquement lorsque que L croît. Il suffit de prendre la valeur minimale de L tel que l'erreur classifieur soit au plus proche du point de convergence. La figure 4 illustre le choix du paramètre L. La seconde étape est le choix de d_{red} sachant L (fixé dans l'étape précédente).

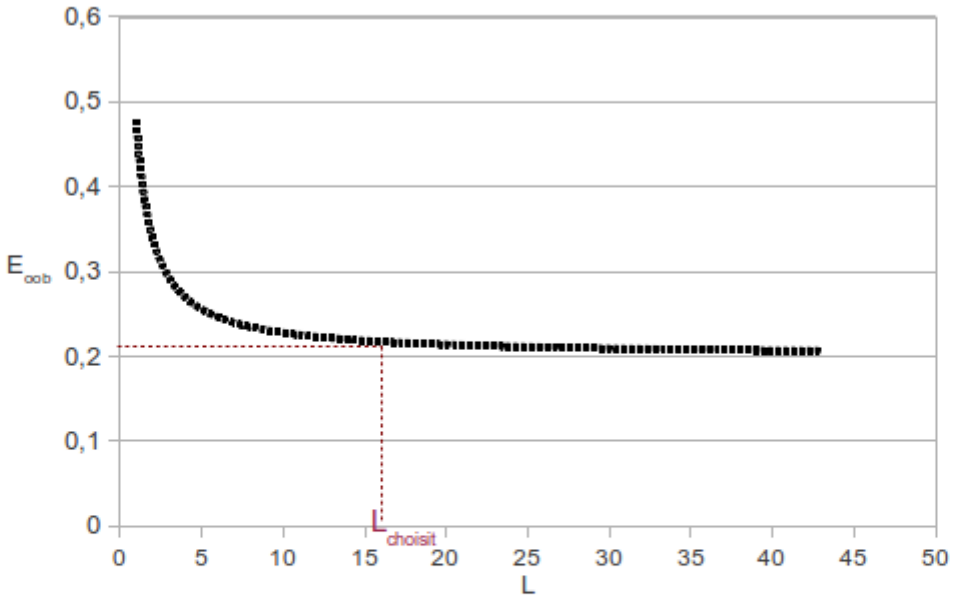

FIGURE 4 – Représentation de l'allure de l'erreur *out-of-bag* pour un nombre fixé de caractéristiques d_{red} en fonction de L.

L'évolution de l'erreur en fonction de d_{red} n'est pas strictement décroissante. Elle passe par un minimum, puis va re-augmenter. Le paramètre d_{red} correspond au minimimum de la fonction erreur. D'après [10], ce minimum est tel que $d_{red} \ll d$.

L'ensemble classifieur de Kodovsky utilise l'analyse du discriminant linéaire de fisher (FLD). Cette méthode permet par réduction de dimensions de maximiser la compacité des classes (*cover* ou *stego*), c'est-à-dire de regrouper lors de projection les images d'une même classe. En pratique, cela équivaut à calculer la matrice de covariance de taille d^2_{red} de chaque vecteur $\mathbf{f}_{red}$ et donc la

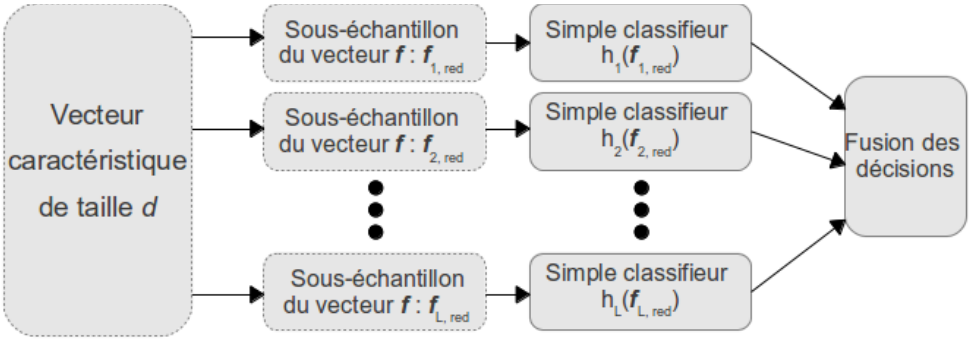

FIGURE 5 – Représentation du fonctionnement de l'ensemble classifieur.

complexité de ce classifieur est $O(d_{red}^2.L.N)$. L'ensemble average perceptron est un ensemble classifieur où chaque classifieur simple est un perceptron. La complexité d'un tel classifieur est alors $O(d_{red}.L.N)$. L'ensemble classifieur est d'après [6] une alternative prometteuse au SVM pour la stéganalyse.

3.3 Comparaisons des principales approches

Comme nous l'avons décrit dans la partie précédente 3.2, les algorithmes d'apprentissage ne peuvent pas, de part leur complexité et mécanismes différents être éxécutés sur la même base d'apprentissage. Sachant que la taille de la base d'apprentissage n'est pas la même, il est difficile de les comparer. Nous allons donc donner quelques résultats de comparaison provenant de [3, 14, 6], mais la comparaison ne sera pas faite entre toutes les approches.

Les résultats que nous présentons ici proviennent de [3, 14]. L'algorithme d'insertion utilisé est nsF5 [8]. Cet algorithme insère les données dans les blocs issus d'une transformation en cosinus discret, qui a lieu lors d'une compression jpeg. Celui-ci est donc particulièrement efficace pour les images jpeg. Nous allons nous placer successivement dans deux cas utilisant des bases de données différentes.

Dans leur article [3], Ivans Lubenko et Andrew D. Ker, récupèrent à l'aide d'un réseau social 800 000 images, toutes compressées identiquement provenant de 200 utilisateurs différents. Cette base de données est très hétérogène, mais les données d'entrainement et de tests, bien que différentes proviennent des mêmes utilisateurs. Nous ne sommes donc pas dans un cas de *cover-source mismath* total [14]. Les résultats présentés dans la figure 6 proviennent de [3]. Sur cette figure, ils représentent l'éxactitude de différents algorithmes en fonction de la taille de la base de données d'apprentissage.

Nous constatons que dans ce scenario, l'algorithme SVM converge très rapidement (avec uniquement 20 000 données d'apprentissage) vers l'asymptote avec 93.5% de prédiction correcte. Toutefois, son utilisation est impossible avec un plus grand jeu de données. Le perceptron converge aussi mais avec beaucoup plus de données d'apprentissage (1.6 million) et oscille très fortement dans ses résultats devenant peu fiables après certaines mises à jour. L'ensemble classifieur de Kodoský converge aussi rapidement que les méthodes SVM vers l'asymptote. Par contre, l'ensemble classifieur permet de traiter de plus grands ensembles de données, car sa complexité est plus faible, pouvant donc potentiellement être plus adapté pour traiter les problèmes de *cover-source mismatch* [14]. La méthode d'ensemble classifieur avec des perceptrons converge également et donne des résultats sensiblement meilleurs que les ensembles FLD ou SVM, cependant il est nécessaire d'effectuer l'apprentissage sur les 1.6 million d'images.

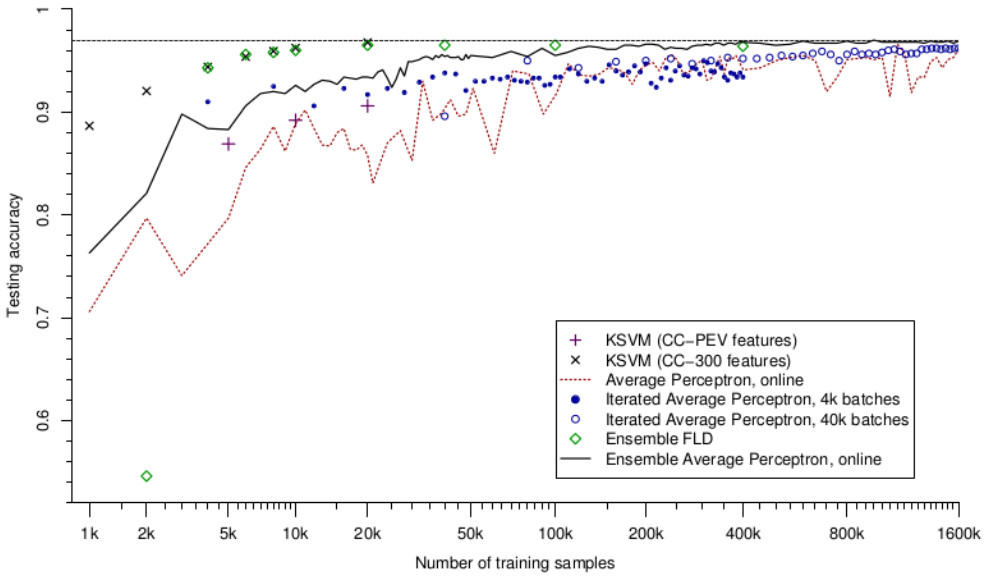

FIGURE 6 – Résultats des expériences avec une charge de 0.1 bpnc (bit par coefficient non nul). Source : [3]

D'aprés la figure 6, qui traite une base d'images hétérogènes, l'ensemble classifieur ne classe pas mieux une image *stego* ou *cover* que l'algorithme plus classique SVM. Nous allons étudier maintenant à une deuxième expérience. Dans [14], Ivans Lubenko et Andrew D. Ker s'interressent au cas du *cover-source mismatch*. Les données testées par les algorithmes sont differentes avec les données d'apprentissage : lors de l'apprentissage le stéganalyste ne connaît pas le type d'image utilisé par le stéganographe. Dans ce scénario,

Tableau 1 – Résultats des classifieurs dans le cas du *cover-source mismatch*. Source : [14]

Algorithme de classification	Exactitude des tests	Nombre de données apprentissage
KSVM	80.9%	6 000
Ensemble classifier FLD	83.6%	20 000
Ensemble Average perceptron	81.2%	1 000 000
Meilleur Ensemble Average perceptron	85.1%	400 000

nous nous rapprochons d'un cas réel de stéganalyse de type analyse automatique de trafic.
Le tableau 1 représente les résultats de plusieurs algorithmes dans le cas d'une insertion plus faible (0.05bpnc avec l'algorithme nsF5). Du fait d'une insertion différente, nous ne pouvons pas comparer directement l'expérience précédente (voir figure 6) avec le tableau 1. Toutefois, comme nous le constatons, dans ce nouveau scenario les algorithmes de type SVM sont nettement moins efficaces en terme d'exactitude que les algorithmes d'ensemble classifieur. Nous constatons même que l'ensemble average perceptron, devient plus interressant que l'ensemble FLD lors de l'apprentissage sur 400 000 données. L'hypothèse impliquant que l'utilisation d'ensemble classifieurs linéaires pour traiter de grandes bases de données et permettant ainsi de mieux lutter face au problème du *cover-source mismatch* [14] semble être prometteuse. Il est donc nécessaire de vérifier cela avec des conditions plus difficiles : images non compressées et insertion plus performante.

Dans ce stage, nous utiliserons l'algorithme HUGO car celui-ci est plus difficilement détectable que nsF5. Dans un autre article [6], les auteurs altèrent sensiblement la base de données BOSSBase [10] contenant 15 000 photos homogènes en y rajoutant 6 500 photos prises par diverses caméras. Ce rajout de données pour l'apprentissage va affaiblir l'efficacité des algorithmes de type SVM et les ensembles classifieurs FLD. Le *cover-source mismatch* est le principal problème pour l'utilisation de la steganalyse en situation réelle [6]. Dans le stage, nous proposerons et expérimenterons des méthodes permettant de lutter contre celui-ci.

10. BOSSBase est la base de données d'images utilisée pour la compétition BOSS. Accès via : http ://exile.felk.cvut.cz/boss/BOSSFinal/Materials/BossRank.zip

4 Stage

4.1 Objectifs généraux du stage

Lors de ce stage, nous nous trouvons dans un contexte où le stéganalyste possède un grand volume d'images. Le but est de vérifier qu'avec un grand volume d'images, le problème du *cover-source mismatch* est mieux traité [14]. Pour analyser et classer chaque image de cette grande base de données, nous allons utiliser des méthodes à complexité linéaire. Nous nous intéressons notamment à une amélioration de l'ensemble classifieur (partie 3.2.3) et du perceptron (partie 3.2.2). Les idées d'amélioration sont proposées dans le chapitre suivant 4.2.

4.2 Contributions envisagées

4.2.1 Pondération des simples classifieurs

Objectif : Dans l'ensemble average perceptron, on constate que chaque classifieur simple a la même influence sur le choix de la décision finale. Nous allons chercher à réduire l'impact des perceptrons émettant le plus de prédictions erronées. À l'inverse, nous voulons augmenter l'impact des perceptrons faisant le plus de prédictions correctes.

Fonctionnement : Chaque classifieur possède une version différente du sous-échantillonage de $\mathbf{f}$. Ce sous-échantillonnage est choisi pseudo-aléatoirement impliquant des performances (en terme de faux positifs et faux négatifs) variant d'un classifieur simple à un autre. L'idée est donc de pondérer chaque classifieur d'un poids α_i avec $i \in \{0..L\}$. Nous pouvons donc modifier la formule de prise de décision de l'ensemble classifieur (équation 5) en rajoutant les poids α_i, avec $i \in \{0..L\}$:

$$S = sign(\sum_{i=0}^{L} (\alpha_i.h_i(\mathbf{f}))) \tag{7}$$

Avec h_i la fonction représentant un classificateur faible, $\mathbf{f}$ le vecteur caractéristique et *sign* la fonction retournant le signe.
Nous allons maintenant décrire la méthode du calcul de chacun des poids α_i. Pour cela il faut introduire deux nouveaux paramètres associés à chaque classifieur : t^v qui comptabilise le nombre de mises à jour où le label $\in \mathscr{L}$ est correctement prédit, et t^f qui comptabilise le nombre de mises à jour où le label n'est pas correctement prédit.
Plusieurs stratégies peuvent être employées pour calculer α. Nous proposons une méthode peu complexe qui consiste à associer un poids proportionnel à la probabilité d'images correctement affectées, définit tel que :

$$\alpha_i = \frac{t_i^v}{t_i^v + t_i^f}.C \tag{8}$$

Avec la constante C permettant de normaliser les poids, c'est-à-dire tel que $\sum_{i=0}^{L} \alpha_i = 1$.
Une autre méthode proposée permettant de limiter plus rapidement l'influence des classifieurs simples peut-être interressante. Celle-ci décrit dans l'équation 9, diminue très rapidement l'influence d'un classifieur faible en fonction du nombre de prédictions inexactes commises, pour cela nous utilisons la fonction exponentielle.

$$\alpha_i = \exp\left(\frac{t_i^v - t_i^f}{t_i^v + t_i^f}\right).C \tag{9}$$

La constante C permet de normaliser les poids, c'est-à-dire tel que $\sum_{i=0}^{L} \alpha_i = 1$.

Ce mécanisme intervient au niveau de l'ensemble classifieur, mais il est également interessant de regarder directement les faibles classifieurs. Nous proposons une méthode d'amélioration des perceptrons dans la section suivante.

4.2.2 Adaptation des vecteurs caractéristiques

Objectif : L'ensemble classifieur attribue à chaque classifieur faible un vecteur caractéristique $\mathbf{f}_{red}$. Nous nous intéressons ici aux classifieurs de type perceptron pour leurs mises à jour progressives (*online*). Nous pouvons facilement, en cours d'apprentissage, retirer une caractéristique de leur vecteur caractéristique $\mathbf{f}_{red}$. Ainsi, nous espérons améliorer le perceptron en retirant du vecteur caractéristique $\mathbf{f}_{red}$ les caractéristiques menant à une contre performance.

Fonctionnement : Notons $\mathbf{f}_{i,red}$ le vecteur caractéristique du classifieur faible $h_i()$ avec $i \in \{0..L\}$. L'ensemble des caractéristiques présentes dans $\mathbf{f}_{i,red}$ ne constitue pas obligatoirement un ensemble discriminant.
Nous proposons dans un premier temps, de nous intéresser à la détection des caractéristiques incompatibles parmi un groupe de caractéristiques. La détection de ces caractéristiques "à enlever" doit être évolutive et sera effectuée uniquement sur les perceptrons peu performants, c'est-à-dire possédant un faible ratio $\frac{t_i^v}{t_i^f + t_i^v}$.
Pour les perceptrons peu performants identifiés, nous proposons de détecter les caractéristiques "faibles". Pour cela, nous suggérons d'analyser pour chaque caractéristique du perceptron peu performant, l'ensemble des perceptrons de l'ensemble classifieur la possédant. Ainsi pour définir la caractéristique "faible", nous comparons l'efficacité de tous les perceptrons possédant cette caractéristique. Nous définissons $\mathscr{H}$ l'ensemble des perceptrons h_i et $\mathscr{H}_{test}$ un sous-ensemble de $\mathscr{H}$. De plus, notons c la caractéristique à

supprimer et c_{test} une caractéristique quelconque appartenant au perceptron peu performant d'indice k dont le vecteur d'apprentissage est $\mathbf{f}_{k,red}$, nous avons alors :

$$c = \underset{c_{test}}{arg\ min} \left(\mathscr{H}_{test} = \{x \in \mathscr{H} | c_{test} \in \mathbf{f}_{x,red}\}, c_{test} \in \mathbf{f}_{k,red}; \frac{1}{|\mathscr{H}_{test}|} \sum_{i \in \mathscr{H}_{test}} \alpha_i \right) \tag{10}$$

Avec $\mathscr{H}_{test}$ l'ensemble contenant tous les perceptrons possédant la caractéristique c_{test}, $|\mathscr{H}_{test}|$ est la cardinalité de l'ensemble $\mathscr{H}_{test}$ et α_i est l'efficacité normalisée d'un perceptron, voir la section précédente : 4.2.1.
Ainsi une caractéristique provoquant le plus d'erreurs dans l'ensemble des perceptrons peut être détectée et *a posteriori* supprimée. L'apprentissage qui suivra pour le perceptron k se fera sur un nouveau vecteur caractéristique $\mathbf{f}_{k,red}$, de dimension inférieur, mais plus efficace pour la séparation linéaire.

4.2.3 Autres perspectives

Selon le temps disponible pour le stage, nous nous intéresserons à d'autres perspectives que nous nous contentons d'énoncer ici.

En s'inspirant des algorithmes génétiques, nous pouvons développer une méthode permettant de renouveler les perceptrons peu performants [11]. Le but est de séléctionner les perceptrons "faibles" avec la même méthode que celle utilisée dans 4.2.2 et de les supprimer. Suite à la suppression, nous allons créer un nouveau percepton possédant les mêmes caractéristiques que le perceptron détruit, mais chaque caractéristique sera obtenue par la moyenne de ces mêmes caractéristiques présentes dans d'autres perceptrons. Cette méthode permet d'acquérir de nouveaux perceptrons travaillant sur des vecteurs caractéristiques uniques issus de la fusion de multiples perceptrons plus efficaces.

Toutes les méthodes présentées précédemment s'intéressent à des classifieurs basés uniquement sur le perceptron. Ici, nous proposons une méthode consistant à changer le classifieur faible de l'ensemble classifieur en un agrégat de classifieurs. Par exemple, chaque classifieur simple peut comporter un perceptron, un réseau de plusieurs perceptrons et un discrimant linéaire de fisher. Le vote de chaque classifieur simple peut être, par exemple, le résultat d'un vote majoritaire interne à chaque agrégat. Cette méthode permet potentiellement d'augmenter la robustesse de chaque classifieur simple et donc celle de l'ensemble classifieur.

4.3 Protocole

Lors de ce stage, nous interprèterons le rôle du stéganalyste. Le but est de proposer une méthode permettant de décider si une image contient un message ou pas. Nous allons également devoir jouer le rôle du stéganographe, afin de générer nos jeux d'apprentissage et de tests composés des images *stego* et *cover*.
Notre étude porte sur une méthode de classification basée sur des algorithmes d'apprentissage. Nous allons avoir besoin d'acquérir une grande quantité d'images (cf. annexe A). Nous en avons déjà récupéré une grande partie, environ 500 000. Puis, à l'aide d'un algorithme d'insertion, décrit dans en section 2.3, un message sera caché dans toutes les images préalablement dupliquées. Enfin nous allons jouer le rôle du stéganalyste, c'est-à-dire extraire pour chaque image, nous avons calculé un vecteur caractéristique (cf. annexe B). La dernière étape est celle de classification proprement dite, en utilisant les différentes méthodes proposées dans la section 4.2. Celles-ci seront comparées aux algorithmes d'apprentissages classiques, tel que : SVM (sur un jeu d'apprentissage réduit), ensemble FLD, ensemble average perceptron. La figure 7 synthétise la totalité de ce processus.
Le protocole étant définie, lors du stage il sera nécessaire de paralléliser le calcul des vecteurs caractéristiques afin de pouvoir le déporter sur le super calculateur hpc@lr [11]. Ensuite, nous implémenterons les méthodes de classification décrites dans la section 3 et nos propositions d'amélioration de la section 4.2. Nous réunirons et comparerons les résultats des différents algorithmes dans le cas d'un *cover-source mismatch* total.

5 Références bibliographiques

Références

[1] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, page 144–152, New York, NY, USA, 1992. ACM.

[2] Marc Chaumont and Sarra Kouider. Steganalysis by ensemble classifiers with boosting by regression, and post-selection of features. In *2012 19th IEEE International Conference on Image Processing (ICIP)*, pages 1133 –1136, October 2012.

11. www.hpc-lr.univ-montp2.fr

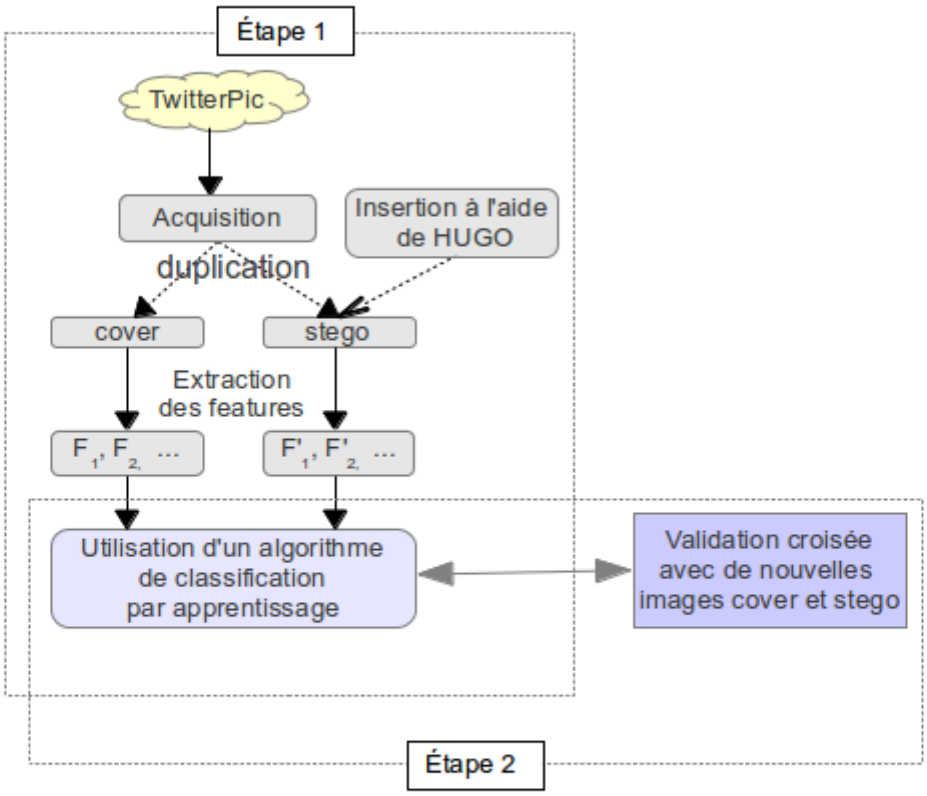

FIGURE 7 – Schéma résumant la méthodologie du protocole expérimental.

[3] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3) :277–296, December 1999.

[4] Jessica Fridrich. *Steganography in Digital Media : Principles, Algorithms, and Applications*. Cambridge University Press, November 2009.

[5] Jessica Fridrich, Miroslav Goljan, and David Soukal. Higher-order statistical steganalysis of palette images. pages 178–190, June 2003.

[6] Jessica Fridrich, Jan Kodovský, Vojtéch Holub, and Miroslav Goljan. Breaking hugo : the process discovery. In *Proceedings of the 13th international conference on Information hiding*, IH'11, pages 85–101, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Jessica J. Fridrich and Jan Kodovský. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, pages 868–882, 2012.

[8] Jessica J. Fridrich, Tomás Pevný, and Jan Kodovský. Statistically undetectable jpeg steganography : dead ends challenges, and opportunities. In *Proceedings of the 9th workshop on Multimedia & security*, MM&Sec '12, pages 3–14, 2007.

[9] Jan Kodovský and Jessica Fridrich. Steganalysis in high dimensions : fusing classifiers built on random subspaces. pages 78800L–78800L, February 2011.

[10] Jan Kodovský, Jessica J. Fridrich, and Vojtech Holub. Ensemble classifiers for steganalysis of digital media. *IEEE Transactions on Information Forensics and Security*, pages 432–444, 2012.

[11] L.I. Kuncheva and L.C. Jain. Designing classifier fusion systems by genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(4) :327 – 336, November 2000.

[12] Ivans Lubenko and Andrew D. Ker. Steganalysis using logistic regression. pages 78800K–78800K, February 2011.

[13] Ivans Lubenko and Andrew D. Ker. Going from small to large data in steganalysis. volume 8303, page 16, February 2012.

[14] Ivans Lubenko and Andrew D. Ker. Steganalysis with mismatched covers : do simple classifiers help ? In *Proceedings of the on Multimedia and security*, MM&Sec '12, pages 11–18, New York, NY, USA, 2012. ACM.

[15] Tomáš Pevný. Detecting messages of unknown length. pages 78800T–78800T, February 2011.

[16] Tomáš Pevný, Patrick Bas, and Jessica Fridrich. Steganalysis by subtractive pixel adjacency matrix. In *Proceedings of the 11th ACM workshop on Multimedia and security*, MM&Sec '09, page 75–84, New York, NY, USA, 2009. ACM.

[17] Tomáš Pevný, Tomáš Filler, and Patrick Bas. Using high-dimensional image models to perform highly undetectable steganography. In Rainer Böhme, Philip W. L. Fong, and Reihaneh Safavi-Naini, editors, *Information Hiding*, number 6387 in Lecture Notes in Computer Science, pages 161–177. Springer Berlin Heidelberg, January 2010.

[18] Hans Georg Schaathun. *Machine Learning in Image Steganalysis*. Wiley - IEEE. Wiley-IEEE Press, 1 edition, October 2012.

Appendices

Dans cette section, nous décrivons le travail réalisé en parallèle de la rédaction du rapport bibliographique et qui a consisté à préparer les données qui seront utilisées dans le stage à savoir acquérir un gros volumes d'images hétérogènes via le réseau social Twitter et leur associer des vecteurs caractéristiques.

A Acquisition des images via le réseau social Twitter

Pour notre étude, nous allons surtout nous intéresser à l'étude d'algorithmes de faible complexité. Toutefois, pour qu'ils fonctionnent correctement, comme vu dans la partie 3.3, il est nécessaire d'avoir un très grand jeu d'apprentissage. Contrairement aux algorithmes de haute complexité, comme les machines à vecteurs de support qui consomment un jeu de données d'environ dix mille images ; les ensembles de perceptrons fonctionnent à une toute autre échelle. En effet, ils auront besoin de plus d'un million d'images pour converger vers un résultat satisfaisant. Outre le problème d'obtenir une grande base de données, la difficulté est d'obtenir une base de données hétérogènes, le but étant de se placer dans un cas de *cover-source mismatch* totale. Pour cela, nous avons récupéré en parallèle de l'étude bibliographique un gros volume d'images provenant de twitter images [12]. Grâce aux utilisateurs de ce réseau social, nous avons obtenu des images très diversifiées : images de synthèses, photographies professionnelles et amateurs, photos de basse et haute qualité, etc... Les images ainsi récupérées sont le plus souvent compressées. Pour notre scénario, nous allons donc les décompresser du format jpeg au format ppm non compressé. A cause de la compression, dont l'intensité varie entre les images, les hautes fréquences ont été lissées et diminuées. Ce phénomène favorise les résultats du stéganalyste.
Twitter images autorise les applications à réaliser 500 requêtes par heure (et par ip) vers leurs serveurs. Par conséquent, nous avons anticipé la récolte de ces gros volumes d'images dès le début de l'étude bibliographique afin de récupérer suffisamment d'images. En effet si nous voulons acquérir deux millions d'images, la durée est d'environ :

$$Temps = \frac{166}{N}(en\ jours) \tag{11}$$

avec N le nombre d'ordinateurs avec des ip différentes que nous possédons.

B Vecteur caractéristique

Afin d'analyser une image, il est nécessaire de la transformer en vecteur. Toutefois, comme montré dans la figure 1 l'histogramme ne peut pas toujours suffire à caractériser une image. La caractérisation d'une image peut se faire dans plusieurs domaines [18] : fréquentiel, ondelettes, spatial. Dans notre étude, nous faisons l'hypothèse que le stéganalyste connaît l'algorithme d'insertion qui est HUGO. Celui-ci insère dans le domaine spatial, voir 2.3. Nous allons donc utiliser des vecteurs caractéristiques dans ce domaine spatial [7] [16]. Dans le modèle SPAM [16], l'idée est d'utiliser des matrices de

12. http ://twitpic.com/

différences de pixels, et donc des filtres passe haut. Une fois l'image filtrée, il est nécessaire de quantifier chaque pixel afin de limiter la taille du vecteur caractéristique. Nous notons T le seuil de quantification, c'est-à-dire que le bruit calculé pour chaque pixel doit être inférieur ou égale à $+T$ et supérieur ou égale à $-T$. Ensuite le vecteur caractéristique sera composé du calcul de la co-occurence des paires de pixels (premier ordre) et des triplets de pixels (deuxième ordre) dans les différentes directions : horizontale, verticale et diagonale. Le nombre de caractéristiques calculées en fonction du seuil T est :

$$\begin{cases} \text{Premier ordre} & : 2.(2.T+1)^2 \\ \text{Deuxième ordre} & : 2.(2.T+1)^3 \end{cases} \tag{12}$$

Par exemple, le modèle avec un $T_{premier\ ordre} = 4$ et $T_{second\ ordre} = 3$ aura 848 caractéristiques, ce modèle s'appelle SPAM-848 [16].

Pour décerner l'algorithme HUGO, le modèle SPAM ne suffit pas, il faut nécessairement utiliser un autre modèle le complétant. Celui-ci [7] regroupe tout un ensemble de sous-modèles permettant d'obtenir différentes relations de bruit entre les voisins. Les sous-modèles se constituent de filtres linéaires et non linéaires. Le code permettant d'obtenir ce vecteur caractéristique est disponible sur le site web [13] de Jessica Fridrich. Toutefois, celui-ci est implémenté uniquement en Matlab, par souci de rapidité et afin de pouvoir l'utiliser sur le super calculateur hpc@lr nous allons le réimplémenter en C.

13. http ://dde.binghamton.edu/download/feature_extractors/