Classification automatique d'images biologiques par variantes d'arbres de décision

Axel Mathéi

Promoteur : Louis Wehenkel

Jury :
Pierre Geurts
Raphaël Marée
Marc Van Droogenbroeck

Année académique 2010-2011
Travail de fin d'études présenté en vue de l'obtention du grade de master en sciences informatiques, à finalité approfondie

Résumé

L'équipe du département « Systèmes et Modélisation » de l'Université de Liège a développé une méthode générique de classification automatique d'images basée sur l'apprentissage automatique et l'applique sur des problèmes concrets dans le domaine biomédical notamment.

Le but de ce travail est de proposer et implémenter plusieurs extensions simples mais consommatrices de ressources de cette méthode, s'inspirant d'approches récentes dans le domaine, et d'étudier leurs performances sur plusieurs catégories d'images.

Remerciements

Je tiens tout d'abord à remercier M. Raphaël Marée pour m'avoir proposé ce sujet et pour avoir suivi avec attention sa réalisation.

Je remercie M. Louis Wehenkel pour avoir accepté d'en être le promoteur, ainsi que MM. Pierre Geurts et Marc Van Droogenbroeck pour avoir accepté de faire partie du jury.

Je tiens aussi à remercier tous mes camarades de classe pour leurs nombreux conseils en tous genres prodigués durant ces 5 dernières années, en particulier Pierre Lepropre et Julien Osmalskyj. Mes études n'auraient pas été aussi enrichissantes sans eux.

Enfin, je remercie bien entendu ma famille pour m'avoir fourni un environnement de travail idéal durant toutes mes études.

Table des matières

Table des figures

Liste des algorithmes

Chapitre 1

Introduction

Grâce aux évolutions technologiques de ces dernières années, on peut maintenant produire facilement et pour un coût modéré des photographies numériques de qualité. Le monde biomédical produit chaque jour une grande quantité d'images, pouvant être aquises de différentes manières : rayons X, résonance magnétique, microscope électronique, etc. Cela entraine chez les scientifiques utilisant ces images le besoin de les classer le plus rapidement et le plus précisément possible, suivant des critères prédéfinis. Malheureusement, ce travail répétitif est encore souvent réalisé par un opérateur humain qui, contrairement à un ordinateur, est soumis à la fatigue et enclin à l'erreur.

Le problème qui nous intéresse ici est la **classification d'images supervisée**.

Supposons que nous soyons face à un immense ensemble d'images à classer. Le principe de la classification supervisée est le suivant : un expert humain classe un échantillon d'images pris dans cet ensemble en plusieurs catégories (sémantiques), ensuite à partir de cet échantillon le programme va construire un modèle puis inspecter toutes les autres images de l'ensemble et assigner à chacune d'elles une des classes définies précédemment, celle qu'il aura jugée la plus probable de contenir l'image suivant le modèle. Ce problème occupe les chercheurs en vision par ordinateur depuis des dizaines d'années.

Ce principe de supervision est opposé à la classification non-supervisée, qui n'a quant à elle besoin d'aucune intervention humaine. C'est le programme qui doit définir lui-même les classes. Ce problème est très différent et n'utilise en général pas les mêmes méthodes que la classification supervisée, ce sujet ne sera donc pas abordé dans ce rapport.

1.1 Objectif

Ce travail ne démarre pas de zéro, loin de là. Le but est ici de rechercher des variantes qui amélioreraient les résultats d'un algorithme de classification automatique d'images existant : celui des Extra-Trees à sous-fenêtres aléatoires (cf. chapitre 4). Cet algorithme a été proposé dans la thèse de doctorat de Raphaël Marée [Mar05], et a été publié en collaboration avec Pierre Geurts, Justus Piater et Louis Wehenkel [MGPW05].

Ce procédé est intégré à un logiciel de classification automatique d'images : PiXiT ©. Ce logiciel est implémenté en Java et commercialisé par la société PEPITe, dont une version d'essai est disponible pour la recherche [1]. Les variantes implémentées dans le cadre de ce projet ont été intégrées dans le code de PiXiT ©. Les tests, très gourmands en ressources, ont été effectués sur la grille informatique du département du GIGA (Groupe Interdisciplinaire de Génoprotéomique Appliquée).

1.2 Organisation du rapport

Nous commencerons par rappeler quelques concepts d'apprentissage automatique nécessaires à la compréhension des techniques utilisées dans ce travail. Ensuite, nous passerons au chapitre abordant la classification d'images en y énonçant le principe ainsi que quelques algorithmes populaires dans ce domaine. La méthode sur laquelle se base ce travail sera détaillée dans le chapitre suivant, suivie par les variantes qui nous intéressent. Nous parlerons ensuite brièvement de l'implémentation, pour passer enfin aux résultats expérimentaux suivis d'une discussion générale en vue de dégager les points importants que nous pourrions retirer de ces résultats.

1. http ://www.montefiore.ulg.ac.be/~maree/pixit.html

Chapitre 2

Concepts d'apprentissage automatique

Avant d'aborder en détails les algorithmes existants dans le domaine de la classification d'images, voici une introduction rapide aux différents concepts qui seront utilisés par la suite.

2.1 Apprentissage supervisé

On dispose de données composées d'une part de variables d'entrée (*inputs*), qui sont mesurées ou prédéfinies, et d'autre part de variables de sortie (*outputs*) qui sont influencées par les variables d'entrées. Notre objectif est d'utiliser des exemples connus pour pouvoir en présence de nouvelles entrées prédire au mieux les sorties, ce procédé est ce qu'on appelle de l'*apprentissage supervisé* [HTF09].

On parle d'un problème de *classification* lorsque les variables de sorties sont symboliques (ou discrètes), à l'inverse si elles sont continues nous sommes face à un problème de *régression*.

Les données (*dataset*) peuvent être représentées sous forme de table, chaque ligne correspondant à un *objet* et chaque colonne étant une *variable* (ou *attribut*) de l'objet. Dans le cadre de ce travail, les objets seront des images et les attributs seront les pixels de ces images.

	Var1	Var2	Var3	Var4	Var5	Var6	Var7
Objet1	-1	5	4	87	98	22	-11
Objet2	0	5	-5	-44	42	17	23
Objet3	1	4	-4	42	78	8	9
Objet4	1	-2	3	22	58	-8	-7

TABLE 2.1: Exemple de dataset quelconque

L'ensemble des objets dont les entrées et les sorties sont connues (dans notre cas les images déjà classées par l'expert humain) forment ce que l'on appelle l'ensemble d'apprentissage (*Learning Set*).

Cet ensemble sera utilisé dans un premier temps lors de la phase d'apprentissage pour construire un modèle. Ce modèle servira lors de la phase de prédiction à classer les objets faisant partie de l'ensemble de test (*Test Set*).

2.2 Plus proches voisins

La technique des plus proches voisins est une des méthodes d'apprentissage supervisé les plus élémentaires.

Soit un ensemble de test de N objets. Pour chaque objet $o_i\,(i = 1, \ldots, N)$ de l'ensemble de test, on mesure la distance entre celui-ci et chaque élément de l'ensemble d'apprentissage. On en déduit ainsi quels sont les k éléments de l'ensemble d'apprentissage les plus proches de o_i, et on lui assigne la sortie majoritaire parmi ces k éléments. Ici, aucun modèle n'est construit.

La mesure de distance la plus fréquemment utilisée est la distance euclidienne.

L'avantage de cette technique est qu'elle ne requiert aucune phase d'apprentissage et qu'il ne faut garder en mémoire que l'ensemble d'apprentissage. Par contre le temps de calcul est énorme : $\mathrm{O}(d.N)$, où d est le nombre d'attributs de chaque objet et N est le nombre d'objets. Il existe tout de même certaines techniques de sélection d'attributs qui permettent de le réduire [HTF09].

2.3 Arbre de décision

Un arbre de décision est une façon très commode de représenter une fonction de classification. Ce procédé a été inventé deux fois, par deux communautés différentes : celle des statisticiens d'une part (CART [BFSO84]), et par celle de l'intelligence artificielle d'autre part (ID3, C4.5 [Qui86]).

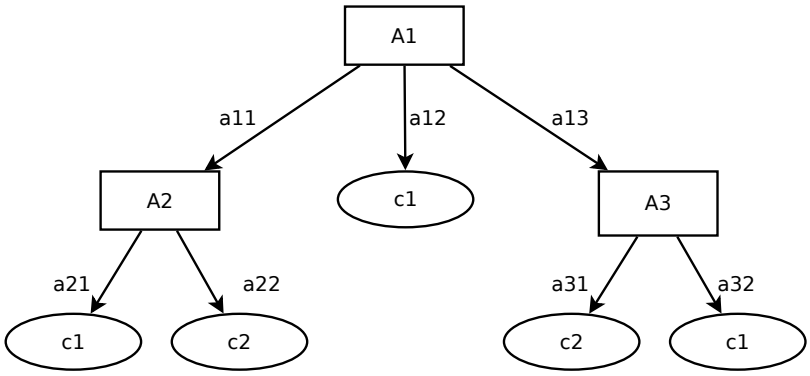


FIGURE 2.1: Exemple d'arbre de décision

Il possède les propriétés suivantes :
– Chaque noeud intérieur teste un attribut.
– Chaque branche correspond à une valeur d'attribut.

– Chaque noeud feuille est labellisé par une classe.

Cette technique a plusieurs avantages :
– Elle est simple à comprendre et à utiliser.
– Le nombre de tests est limité par le nombre d'attributs.
– L'algorithme de construction de l'arbre est efficace.

Arrêtons-nous sur le dernier point pour détailler l'algorithme de construction d'un arbre de décision, tiré de [Weh10] :

Algorithme 2.1 Construction d'un arbre de décision

Algorithme de création learn_dt(LS) :
– Si tous les objets d'un LS (Learning Set) ont la même classe, on crée une feuille avec cette classe.
– Sinon,
  – On cherche l'attribut qui sépare le LS au mieux.
  – On crée un noeud de test pour cet attribut.
  – Pour chaque valeur a de A,
    – On construit $LS_a = \{o \in LS \mid A(o) \, est \, a\}$
    – On relance learn_dt sur LS_a pour créer un sous-arbre à partir de là.

Le point central de l'algorithme est le choix du meilleur attribut, celui qui sépare au mieux le LS. On veut en effet minimiser la taille de l'arbre, et donc maximiser la séparation des classes à chaque étape. Cela équivaut à rendre les successeurs aussi pures que possible. Afin de mesurer la différence de pureté entre un noeud père et ses fils, on utilise la notion d'impureté [Weh10].

Impureté Soit LS un ensemble d'échantillons, et p_j la proportion d'objets de classe j ($j = 1, \ldots, J$) dans LS.

La mesure d'impureté I(LS) satisfait les affirmations suivantes :
– I(LS) est minimale quand $p_i = 1$ et que $p_j = 0$, $\forall \, j \neq i$.
– I(LS) est maximale quand $p_j = 1/J$
– I(LS) est symétrique par rapport à $p_1, \ldots, p_j$

Quelques exemples de mesures d'impuretés courantes :
– Entropie de Shannon : $I(LS) = -\sum_j p_j \log(p_j)$
– L'index Gini : $I(LS) = \sum_j p_j(1 - p_j)$
– Taux d'erreur de classification : $I(LS) = 1 - \max_j p_j$

La meilleure séparation est donc celle qui maximise la réduction d'impureté :

$$\triangle I(LS, \, A) = I(LS) - \sum_a \frac{|LS_a|}{|LS|} I(LS_a)$$

où LS_a est le sous-ensemble des objets de LS tel que $A = a$.

Un noeud sera développé jusqu'à ce qu'il soit assez pur, ou qu'il ait atteint une taille assez réduite (n_{min}).

2.4 Ensemble d'arbres

L'idée ici est de construire plusieurs arbres de décisions différents à partir d'un même ensemble d'apprentissage, et d'aggréger les résultats des prédictions de chaque arbre pour en déduire la prédiction finale.

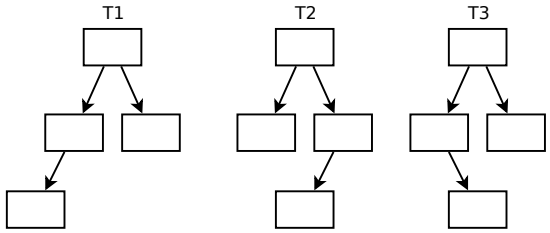


FIGURE 2.2: Exemple d'ensemble d'arbres

Le principal avantage d'un ensemble d'arbres par rapport à un arbre est qu'il est plus stable. Cela se traduit par une réduction de la variance du résultat, c'est-à-dire que pour un problème donné, le résultat variera moins d'un ensemble d'apprentissage à l'autre.

2.4.1 Bagging

Le Bagging (bootstrap aggregating) consiste à construire chaque arbre de l'ensemble à partir d'un sous-ensemble chaque fois différent de l'ensemble d'apprentissage. Ces sous-ensembles sont construit aléatoirement, par tirage avec remise.

2.4.2 Random Forests

Les Random Forests sont construits sur le même principe que le bagging, mais pendant la construction de l'arbre chaque noeud va se limiter à un sous-ensemble des attributs. Le choix de la meilleure séparation ne se fera donc plus sur tous les attributs comme dans un arbre classique mais sur un échantillon de ceux-ci. Cette méthode est logiquement plus rapide que le Bagging, et donne de meilleurs résultats [Bre01].

2.4.3 Extra-Trees

Les Extra-Trees [GEW06] (extremely randomized trees) sont un ensemble d'arbres de décision construits comme expliqué ci-dessus (cf. section 2.3). Ils diffèrent cependant des autres méthodes d'ensembles sur plusieurs points :

- La séparation qui a lieu dans les noeuds internes se fait aléatoirement : les attributs testés et les seuils sont choisis aléatoirement.
- Les Extra-Trees utilisent tout l'ensemble d'apprentissage pour construire les arbres, et plus seulement une partie comme dans la méthode du Bagging (cf. section 2.4.1).

Les Extra-Trees étant à la base de la méthode utilisée dans ce travail, développons plus en détails l'algorithme de séparation des noeuds (cf. algorithme 2.2).

Algorithme 2.2 Séparation d'un nœud d'Extra-Tree (pour des attributs numériques) [GEW06]

Split_a_node(S)
Entrée : le sous-ensemble d'apprentissage S correspondant au noeud à séparer
Sortie : une séparation [a < a_c] ou rien
– Si **Stop_split**(S) est VRAI alors on ne renvoie rien.
– Sinon on sélectionne K attributs $\{a_1, ..., a_K\}$ parmi tous les attributs non constants de S ;
– On génère K séparations $\{s_1, ..., s_K\}$, où s_i = **Pick_a_random_split**(S, a_i), $\forall i = 1, ..., K$;
– On renvoie une séparation s_* tel que Score(s_*, S) = $max_{i=1,...,K}$Score(s_i, S).
Pick_a_random_split(S, a)
Entrées : un sous-ensemble S et un attribut a
Sortie : une séparation
– Soient a^S_{max} et a^S_{min} respectivement les valeurs maximum et minimum de a dans S ;
– On sélectionne un seuil aléatoire a_c, tiré uniformément dans l'intervalle $[a^S_{max}, a^S_{min}]$;
– On renvoie la séparation $[a < a_c]$.
Stop_split(S)
Entrée : un sous-ensemble S
Sortie : un booléen
– Si $|S| < n_{min}$, alors on renvoie VRAI ;
– Si tous les attributs sont constants dans S, alors on renvoie VRAI ;
– Si la sortie est constante dans S, alors on renvoie VRAI ;
– Sinon, on revoie FAUX.

2.5 Machine à support vectoriel

Les SVMs (Support Vector Machine) sont une généralisation des classificateurs linéaires. Ils tentent de maximiser la distance entre la frontière de séparation des objets et les échantillons les plus proches. Les objets n'étant pas toujours linéairement séparables, les SVMs essaient de trouver un espace de plus grande dimension où les objets seraient séparables, ils augmentent ainsi la dimension de l'espace des objets de départ en appliquant une fonction noyau.

La bibliothèque de fonctions utilisée dans le cadre de ce travail implémentant les SVMs est LIBSVM [1].

1. http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Chapitre 3

Classification d'images

Dans ce chapitre, nous allons d'abord rappeler le principe de la classification d'images, puis nous examinerons les algorithmes existants dans ce domaine.

3.1 Principe

La classification d'images supervisée se décompose en 2 phases : la création du modèle et la prédiction utilisant ce modèle.

3.1.1 Création du modèle

Pour créer un modèle, le programme a besoin d'images préalablement classées. C'est donc un opérateur humain compétent qui est chargé de classer manuellement un échantillon d'images suffisamment représentatif. Le modèle sera construit uniquement à partir de ces images comme illustré à la figure 3.1. Cette étape est cruciale, il sera en effet impossible d'obtenir de bons résultats avec un mauvais ensemble d'apprentissage.

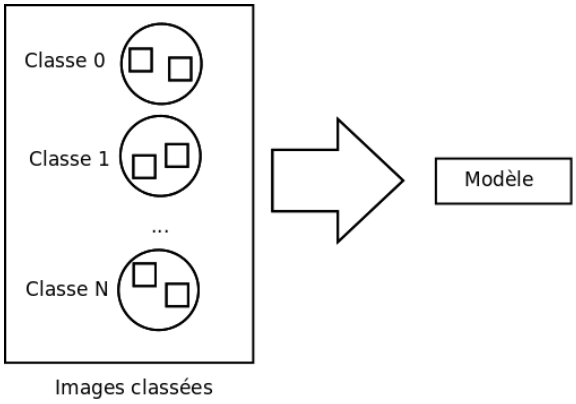


FIGURE 3.1: Création du modèle

3.1.2 Prédiction

Une fois le modèle construit, on peut lui soumettre des images non classées comme le montre la figure 3.2. Le programme va alors se servir du modèle pour voir à quelle classe l'image soumise a le plus de chances d'appartenir. Finalement, pour chaque image inconnue soumise le programme sortira comme réponse la classe la plus probable suivant le modèle.

FIGURE 3.2: Prédiction grâce au modèle

3.2 Algorithmes populaires

Nous allons maintenant passer en revue différentes techniques parmis les plus populaires utilisées pour la classifications d'images.

3.2.1 Plus proches voisins

L'algorithme des plus proches voisins (cf. section 2.2) est souvent considéré comme méthode de base pour la classification d'images. Une approche fréquente est de redimensionner les images vers une taille fixe (par exemple 32x32), éventuellement de les filtrer, et d'utiliser une distance euclidienne.

Néanmoins, une équipe de chercheurs a tout de même réussi à développer un classificateur compétitif uniquement basé sur les plus proches voisins [OBI08]. Leur méthode se distingue sur 2 aspects : d'une part ils utilisent des descripteurs d'images (tels SIFT (Scale-invariant feature transform) ou GB (Geometric-Blur)) mais sans effectuer de sélection parmi ceux-ci (comme c'est le cas dans les méthodes construisant des dictionnaires de descripteurs), d'autre part ils ne considèrent pas des distances image-image mais image-classe. Ils arrivent ainsi à des résultats comparables aux

algorithmes les plus performants du moment sur les bases de données Caltech-101, Caltech-256 et Graz-01. Cela tout en conservant les avantages d'un algorithme non paramétrique : pas de phase d'apprentissage, une capacité à prendre en compte naturellement un grand nombre de classes, et aucun paramètres à optimiser (cela évite donc les risques de surapprentissage).

3.2.2 Machine à support vectoriel et bags of words

Les SVMs et leurs applications à la classification d'images ont fait l'objet de nombreuses recherches ces dernières années [KS07, YLF07, JZS07], et comptent parmis les meilleurs classificateurs existants [MMvdW07, ABM07].

Le point faible de certaines techniques (telles que les plus proches voisins) est qu'elles ont du mal à gérer un nombre important de dimensions, alors que les SVMs sont très performant sans connaissances à-priori du problème, et ce même lorsque le nombre de dimensions de l'espace des variables d'entrée est très grand. Cela est bien illustré par [OCV99], où les auteurs classent des images grâce aux SVMs en se basant uniquement sur des histogrammes couleurs de grande dimension.

Il existe une méthode de classification très populaire basée sur les SVM consistant à les appliquer à des caractéristiques extraites des images à partir de points d'intérêts regroupés en mots visuels (*Bag-of-visual words*) à l'aide, par exemple, de K-Means. Cette méthode a été introduite pour la classification d'images par [CDF+04].

3.2.3 Les réseaux à convolution

Les réseaux à convolution sont des réseaux de neurones qui comportent une fenêtre glissante correspondant à un champ de vision restreint de l'image.

Un neurone d'une couche d'un perceptron est connecté à tous les neurones de la couche précédente, tandis que pour un réseau de neurones à convolution un neurone est connecté à un sous-ensemble de neurones de la couche précédente[PVG01]. Chaque neurone peut être vu comme une unité de détection d'une caractéristique locale, comme illustré à la figure 3.3.

Ce type de réseau a été appliqué avec succès à la reconnaissance de caractères manuscrits isolés [LB94].

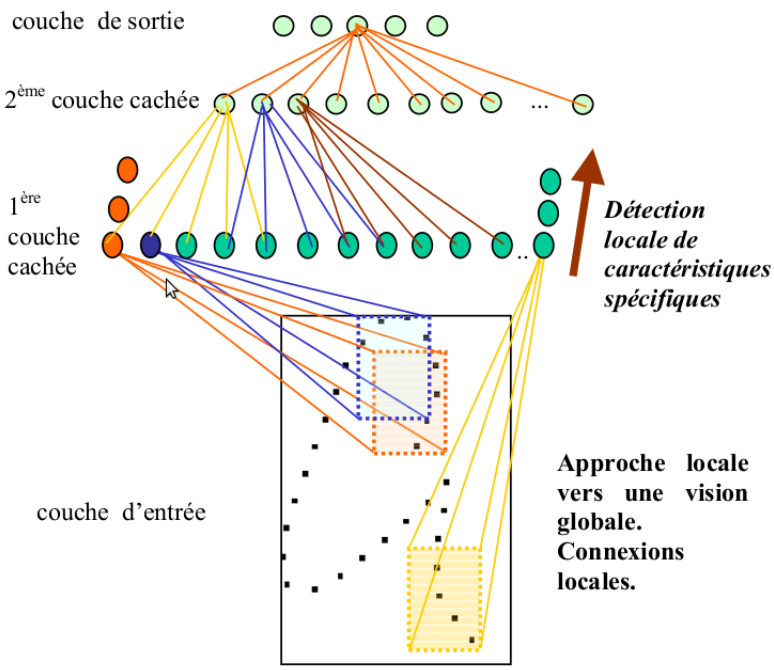


FIGURE 3.3: Réseaux à convolution [PVG01]

3.2.4 Méthodes à base d'arbres

Les arbres de décision ne sont pas en tant que tels compétitifs vis-à-vis d'autres techniques d'apprentissage à cause de leur variance élevée, par contre les ensembles d'arbres réduisent cette variance et sont ainsi à la base de nombreux algorithmes récents.

Les articles [RMW03, MGPW05] développent une comparaison intéressante entre les différentes techniques basées sur les arbres de décisions (arbre de décision classique, Bagging, Random Forests, Boosting, Extra-Trees) et les SVMs. Les 4 bases de données d'images sur lesquelles les tests ont été effectués sont : MNIST (chiffres manuscrits), ORL (visages), COIL-100 (objets en images de synthèse) et OUTEX (textures). Il apparait que l'algorithme s'en sortant le mieux sur ces 4 bases de données de natures très différentes est celui des **Extra-Trees à sous-fenêtres aléatoires**. Cet algorithme a donc à première vue un certain potentiel de généralisation. Les Extra-Trees à sous-fenêtres aléatoires sont la base des variantes développées dans ce travail, et leur fonctionnement sera détaillé dans le chapitre 4.

Vincent Lepetit et Pascal Fua de l'École Polytechnique Fédérale de Lausanne (EPFL) ont développé une technique d'arbres aléatoires (*Random Trees*) permettant de créer un modèle destiné à la détection et au suivi d'objets en temps réel [LF06]. Les tests dans les noeuds des Random Trees sont basés sur la différence d'intensité entre 2 pixels pris au hasard dans le voisinage d'un point d'intérêt.

Frank Moosmann et Frederic Jurie ont introduit une méthode basée sur des arbres aléatoires inspirée des Extra-Trees à sous-fenêtres aléatoires : les *Extremely Randomized Clustering Forests* [MNJ08]. Ils ne sont pas utilisés pour classer directement les fenêtres mais pour les organiser en « mots visuels ». Une méthode similaire basée sur des « mots visuels » (*Bags of features*) sera décrite plus en détails à la section 4.2.

Chapitre 4

Extra-Trees à sous-fenêtres aléatoires

Nous allons maintenant détailler la méthode servant de base à ce travail, à savoir les Extra-Trees à sous-fenêtres aléatoires. Ce résumé de la méthode se base sur les articles [MGPW05, MSG10].

4.1 Classification directe

Étant donné un ensemble d'images classées, une méthode de classification automatique d'images se constitue de 2 phases distinctes :

1. Lors de la phase d'entraînement, elle construit un modèle.
2. Lors de la phase de prédiction, elle se sert de ce modèle pour pourvoir prédire le plus précisément possible la classe d'une nouvelle image non-classée.

Nous allons décrire ici l'utilisation la plus classique des Extra-Trees pour la classification d'images : la classification directe. Cette méthode correspond au mode « C » du logiciel.

4.1.1 Phase d'entraînement

Extraction de sous-fenêtres

La phase d'entraînement consiste d'abord à extraire un grand nombre de sous-fenêtres carrées de tailles et de positions aléatoires des images de l'ensemble d'apprentissage. Ces sous-fenêtres permettent une description à la fois locale et globale de l'image. Ces fenêtres sont ensuite redimensionnées à une taille fixe (16 x 16 dans notre cas), cela permet une plus grande robustesse aux changements d'échelle (c'est-à-dire de décrire de la même façon une même image qui aurait été redimensionnée). Ces sous-fenêtres sont donc décrites par les valeurs de leurs pixels et étiquettées par la classe de l'image dont elles sont tirées (cf. figure 4.1).

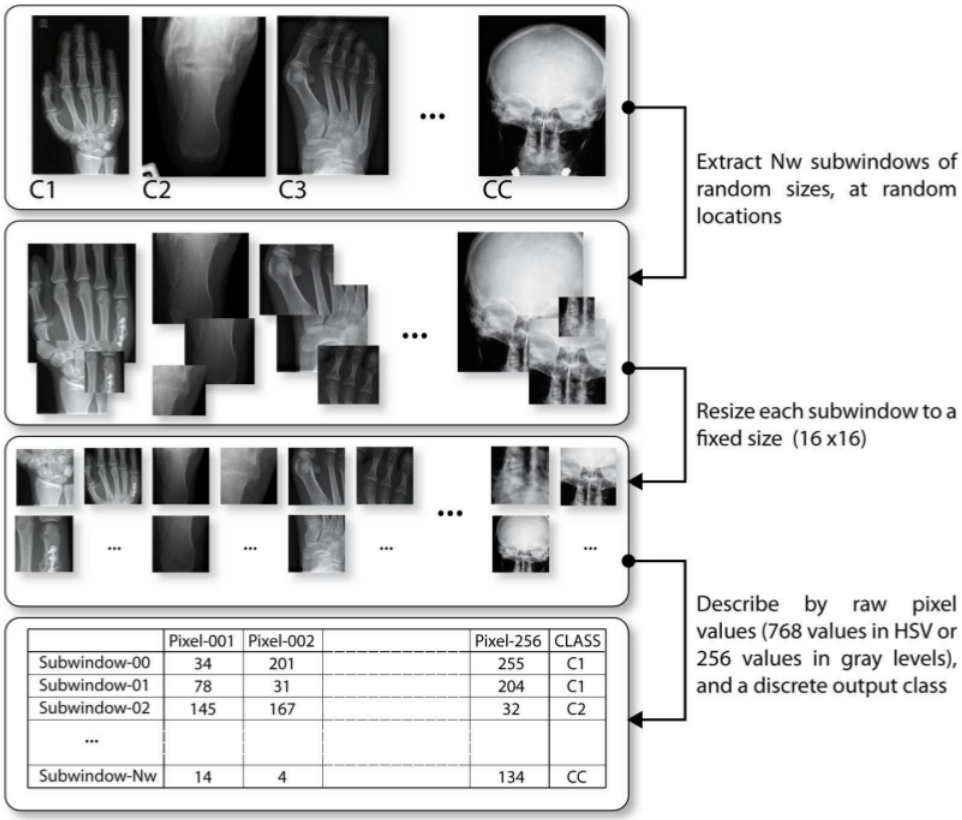


FIGURE 4.1: Extraction des sous-fenêtres [Mar09]

Extra-Trees

Un modèle de classification des sous-fenêtres est alors construit par un ensemble d'arbre de décision extrêmement aléatoires (Extra-Trees) (cf. section 2.4.3). L'algorithme développe T arbres de décisions selon l'algorithme 2.2. Le principe de construction d'un arbre est le suivant : à la racine de l'arbre, on trouve toutes les sous-fenêtres extraites des images de l'ensemble d'apprentissage.

Nous allons maintenant détailler la séparation des noeuds, celle-ci jouant un rôle important dans les variantes présentées par la suite.

Ici dans le cadre de la classification de sous-fenêtres, les tests effectués à l'intérieur des noeuds comparent la valeur d'un pixel fixé (une intensité en niveau de gris, ou une composante d'une couleur) de la sous-fenêtre avec une valeur seuil. Le paramètre k correspond au nombre de pixels choisi aléatoirement à chaque noeud, il est donc compris entre 1 et le nombre de pixels de la sous-fenêtre (dans notre cas : $16 * 16 = 256$). Pour chacun de ces k attributs, une valeur seuil de pixel est choisie aléatoirement (cf. figure 4.2). Le score de chaque test est alors calculé sur le sous-ensemble de sous-fenêtres selon un critère de gain d'information (cf. section 2.3, sous-section Impureté), et le

meilleur des k tests est choisi pour séparer le noeud. Cette procédure est appliquée récursivement jusqu'à ce que l'arbre soit entièrement développé (c'est-à-dire que ses noeuds soient assez purs ou qu'ils aient tous atteints une taille inférieure ou égale à n_{min}). À chaque feuille est associé un vecteur de fréquences, indiquant pour chaque classe la proportion de sous-fenêtres issues d'images cette classe tombées dans cette feuille par rapport au nombre total de sous-fenêtres tombées dans cette feuille. Les T arbres de l'ensemble sont ainsi construits et sauvegardés pour la phase de prédiction.

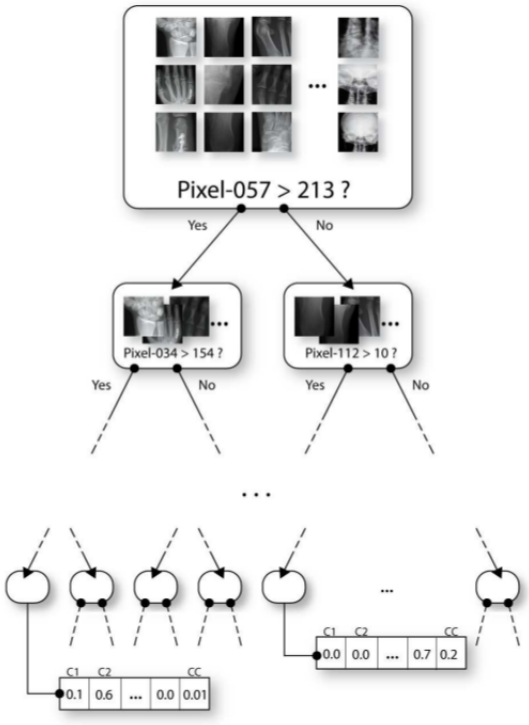


FIGURE 4.2: Séparation des noeuds [Mar09]

4.1.2 Phase de prédiction

Une fois le modèle construit, on peut soumettre au programme des images non labellisées afin de les classer automatiquement.

Pour chaque image I_Q soumise, on procède comme suit (cf. figure) :
- On extrait des sous-fenêtres de I_Q comme à la phase d'entrainement (cf. section 4.1.1). Le nombre de sous-fenêtres extraites pour la phase de prédiction peut être différent du nombre de sous-fenêtres extraites lors de la phase d'entrainement, cela est à préciser aux arguments du programme.
- Ensuite, ces sous-fenêtres sont propagées dans les arbres du modèle.

– Chaque sous-fenêtre se verra alors attribuer le vecteur de fréquences correspondant à la feuille dans laquelle elle est tombée.
– Pour chaque classe, on prend la moyenne des probabilités qu'elle a obtenue pour chaque sous-fenêtre. Cette moyenne correspondra au score de l'image I_Q pour cette classe.
– On attribue finalement à l'image I_Q la classe ayant obtenu le meilleur score.

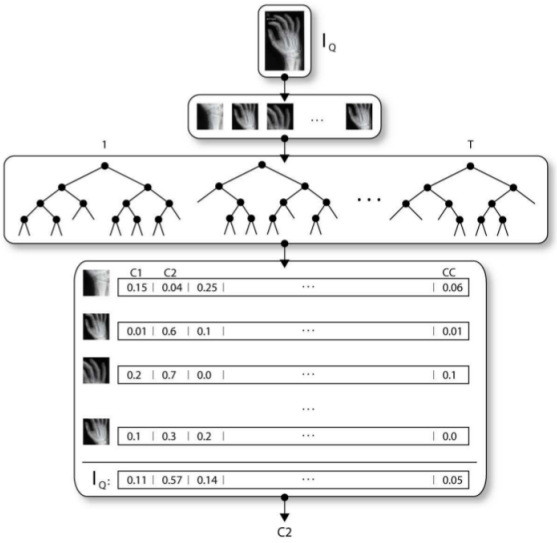


FIGURE 4.3: Prédiction en classification directe [Mar09]

4.2 Bags of Features

Il existe une autre façon d'utiliser les Extra-Trees que la classification directe : ce sont les Bags of Features, proposé par [MNJ08]. Cette méthode correspond au mode « BAGS » du logiciel.

4.2.1 Phase d'entraînement

La construction des Extra-Trees se fait exactement de la même manière que dans la classification classique. La différence réside dans l'exploitation des informations que les Extra-Trees vont nous fournir.

Reprenons l'algorithme d'entraînement depuis le début (cf. figure 4.4) :
– On commence par extraire des sous-fenêtres aléatoires redimensionnées des images de l'ensemble d'apprentissage, exactement comme pour une classification directe.
– Les Extra-Trees sont construits à partir de l'ensemble des sous-fenêtres extraites à l'étape précédente, de la même manière qu'en classification directe.
– Ensuite, chaque image I_i est décrite par un vecteur contenant pour chaque feuille des arbres la proportion de sous-fenêtres appartenant à I_i contenue dans cette feuille. La classe de l'image I_i est ajoutée à la fin du vecteur comme étant la sortie (*output*).

– Ces vecteurs servent finalement à entraîner une machine à support vectoriel (SVM).
Les arbres et la machine à support vectoriel doivent donc tous les deux être sauvegardés pour conserver le modèle, les images de l'ensemble d'apprentissage et les vecteurs ayant entraîné la machine à support vectoriel ne sont plus utiles pour la prédiction.

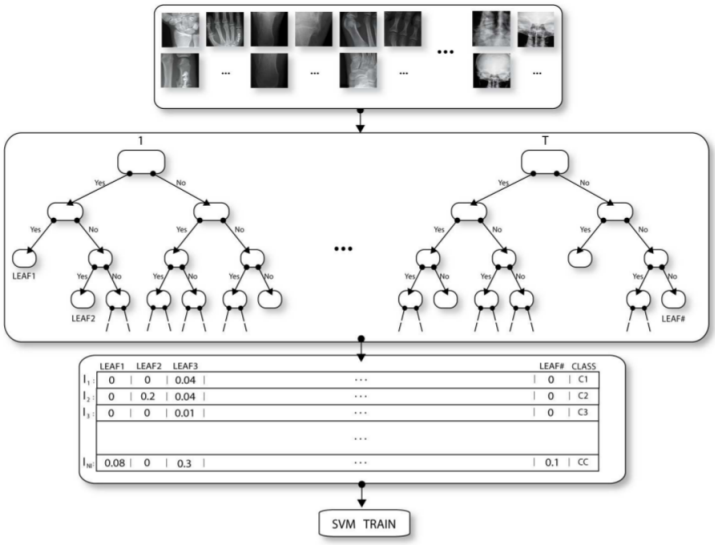


FIGURE 4.4: Entraînement en mode BAGS [Mar09]

4.2.2 Phase de prédiction

La prédiction se fait dès lors assez naturellement comme suit (cf. figure 4.5) :
– On extrait des sous-fenêtres aléatoires, que l'on redimensionne, de l'image soumise I_Q.
– Ces sous-fenêtres sont propagées dans les arbres du modèle entraîné.
– Après propagation, l'image peut donc être décrite par un vecteur contenant pour chaque feuille des arbres du modèle la proportion de sous-fenêtres de I_Q contenue dans cette feuille.
– Ce vecteur est alors passé à la machine à support vectoriel du modèle, qui déterminera à quelle classe appartient l'image.

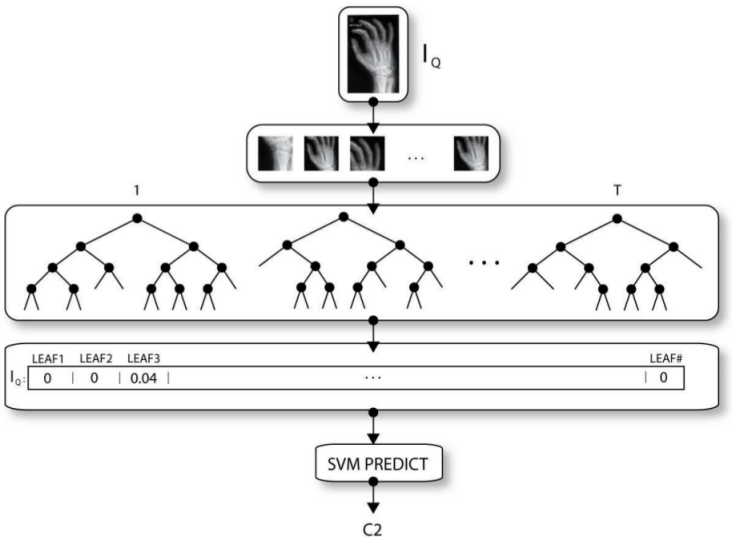


FIGURE 4.5: Prédiction en mode BAGS [Mar09]

Chapitre 5

Variantes

Nous entamons ici la partie originale du travail, à savoir les variantes des Extra-Trees qui ont été testées. Celles-ci interviennent à plusieurs niveaux.

Tout d'abord nous présenterons les variantes de tests au niveau des noeuds dans la section 5.1. Ces différent tests modifient la façon dont les arbres sont construits.

Ensuite, nous verrons dans la section 5.2 les variantes qui interviennent lors de l'extraction des sous-fenêtres. On appliquera ici des transformations géométriques simples (rotations et flips) aux sous-fenêtres extraites.

Enfin, le dernier type de variantes consiste à créer plusieurs versions filtrées des images de l'ensemble d'apprentissage avant de lancer le programme sur celles-ci, cela sera détaillé dans la section 5.3.

5.1 Au niveau des noeuds

Les variantes qui suivent modifient la façon dont les noeuds de l'Extra-Tree sont séparés.

5.1.1 Tests simples

SIMPLETHRES

SIMPLETHRES est le nom donné au test par défaut. Si on reprend l'algorithme 2.2 de développement d'un noeud dans un Extra-Tree, la méthode Pick_a_random_split(S, a) est adaptée à notre situation de classement automatique d'images par :

Algorithme 5.1 Séparation d'un nœud d'Extra-Tree (pour des pixels)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de p dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p < p_c]$.

DIFF

Ici, le pixel p ne sera plus le seul à intervenir, on considèrera la valeur de la différence entre la valeur de p et celle de son voisin directement à sa gauche. Intuitivement, on s'attend ainsi à ce que ce type de test capture les contours fortement marqués de l'image lors de la classification.

Algorithme 5.2 Séparation d'un nœud d'Extra-Tree (DIFF)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit $p_{i,j+1}$ le voisin directement à la droite de p ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $p - p_{i,j+1}$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p - p_{i,j+1} < p_c]$.

DIFFABS

Dans cette variante, on prend la valeur absolue de la différence entre la valeur de p et celle de son voisin directement à sa gauche. Tout comme pour le test DIFF, nous pensons que ce test détectera les contours de l'images et en tiendra donc compte pour la classification.

Algorithme 5.3 Séparation d'un nœud d'Extra-Tree (DIFFABS)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit $p_{i,j+1}$ le voisin directement à la droite de p ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $|p - p_{i,j+1}|$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[|p - p_{i,j+1}| < p_c]$.

DUOPIX

Ce test est similaire au précédent, à la différence qu'on considère ici la somme de p et de son voisin directement à sa gauche. Il n'est plus ici question de détecter les contours fortement marqués, on prend juste en compte une zone de 2 pixels au lieu d'un seul.

Algorithme 5.4 Séparation d'un nœud d'Extra-Tree (DUOPIX)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit $p_{i,j+1}$ le voisin directement à la droite de p ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $p + p_{i,j+1}$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p + p_{i,j+1} < p_c]$.

LOGSIMPLEPIX

Cette variante consiste simplement à prendre les logarithmes des valeurs des pixels (auquelles on ajoute 1, puisque les valeurs des pixels en niveaux de gris sont comprises entre 0 et 255, et que $\log(0) = -\infty$) au lieu des valeurs telles quelles.

Pick_a_random_split(S, p) devient alors :

Algorithme 5.5 Séparation d'un nœud d'Extra-Tree (LOGSIMPLEPIX)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $\log(p + 1)$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[\log(p + 1) < p_c]$.

Le logarithme (cf. figure 5.1) ne change pas la relation d'ordre qui existe entre les valeurs des pixels, mais donne un plus large intervalle de valeurs aux pixels sombres (de valeurs faibles).

5.1.2 Tests aléatoires

Ces tests introduisent un nouveau facteur aléatoire par rapport aux tests simples. Ce facteur aléatoire supplémentaire fait partie du modèle, et doit donc être sauvegardé (ce point concernant l'implémentation sera développé au chapitre 6).

DIFFRAND

Ce test est similaire à DIFF car on prend la différence entre 2 pixels, mais ici on ne prend plus forcément 2 pixels voisins car le deuxième pixel est également choisi aléatoirement.

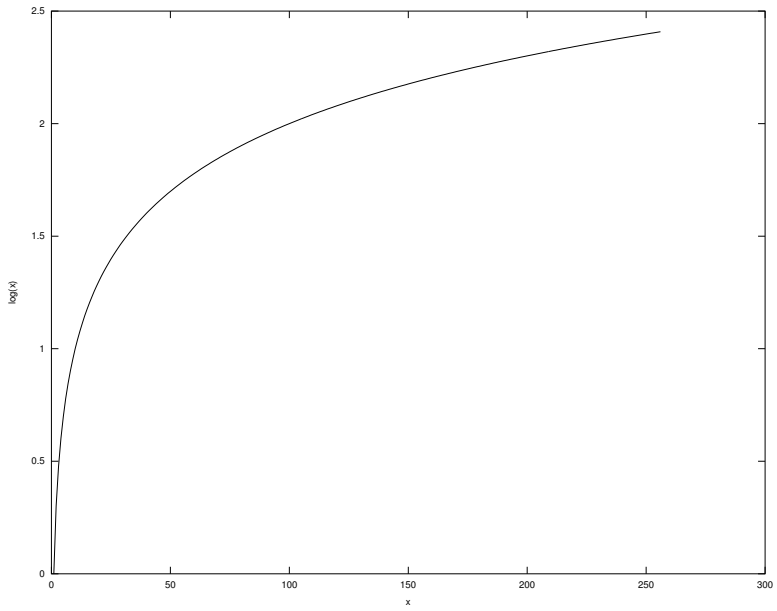


FIGURE 5.1: La fonction logarithme en base 10

Algorithme 5.6 Séparation d'un nœud d'Extra-Tree (DIFFRAND)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit p_{rand} un pixel choisi aléatoirement ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $p - p_{rand}$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p - p_{rand} < p_c]$.

DIFFABSRAND

Ce test est similaire à DIFFRAND : on considère ici la valeur absolue de la différence entre 2 pixels, le deuxième pixel étant également choisi aléatoirement.

Algorithme 5.7 Séparation d'un nœud d'Extra-Tree (DIFFABSRAND)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit p_{rand} un pixel choisi aléatoirement ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $|p - p_{rand}|$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[|p - p_{rand}| < p_c]$.

DUOPIXRAND

Ce test est similaire à DUOPIX car on prend la somme de 2 pixels, mais ici on ne prend plus forcément 2 pixels voisins car le deuxième pixel est choisi aléatoirement.

Algorithme 5.8 Séparation d'un nœud d'Extra-Tree (DUOPIXRAND)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit p_{rand} un pixel choisi aléatoirement ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $p + p_{rand}$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p + p_{rand} < p_c]$.

DIFFTWOPIXLBP

Ici on prend la différence entre le pixel et un de ses 8 voisins directs choisi aléatoirement (cf. figure 5.2).

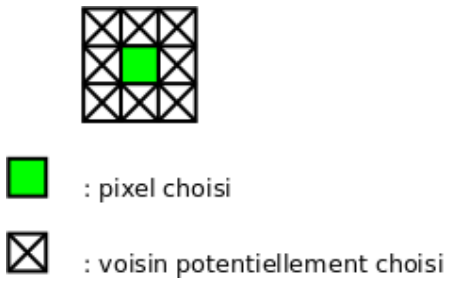


FIGURE 5.2: Voisinage considéré par DIFFTWOPIXLBP

D'une certaine façon, ce type de test imite les descripteurs LBP (Local Binary Patterns [OPM01]) dans un rayon restreint de 1 pixel autour du pixel choisi.

Algorithme 5.9 Séparation d'un nœud d'Extra-Tree (DIFFTWOPIXLBP)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Soit $p_{randNeighbor}$ un pixel choisi aléatoirement parmi les 8 voisins directs de p ;
– Soient p^S_{max} et p^S_{min} respectivement les valeurs maximum et minimum de $p - p_{randNeighbor}$ dans S ;
– On sélectionne un seuil aléatoire p_c, tiré uniformément dans l'intervalle $[p^S_{max}, p^S_{min}]$;
– On renvoie la séparation $[p - p_{randNeighbor} < p_c]$.

RANDOM

Ce test choisi aléatoirement quel type de test appliquer parmi les 9 présentés précédemment.

Algorithme 5.10 Séparation d'un nœud d'Extra-Tree (RANDOM)

Pick_a_random_split(S, p)
Entrées : un sous-ensemble S et un pixel p situé à la ligne i et à la colonne j des sous-fenêtres.
Sortie : une séparation
– Choisit aléatoirement une méthode Pick_a_random_split(S, p) parmi les variantes SIMPLETHRES, LOGSIMPLEPIX, DIFF, DIFFRAND, DIFFABS, DIFFABSRAND, DUOPIX, DUOPIXRAND, DIFFTWOPIXLBP ;
– Renvoie la séparation Pick_a_random_split(S, p) correspondante.

Chaque noeud de l'arbre peut donc être divisé par un test de type différent, comme l'illustre la figure 5.3.

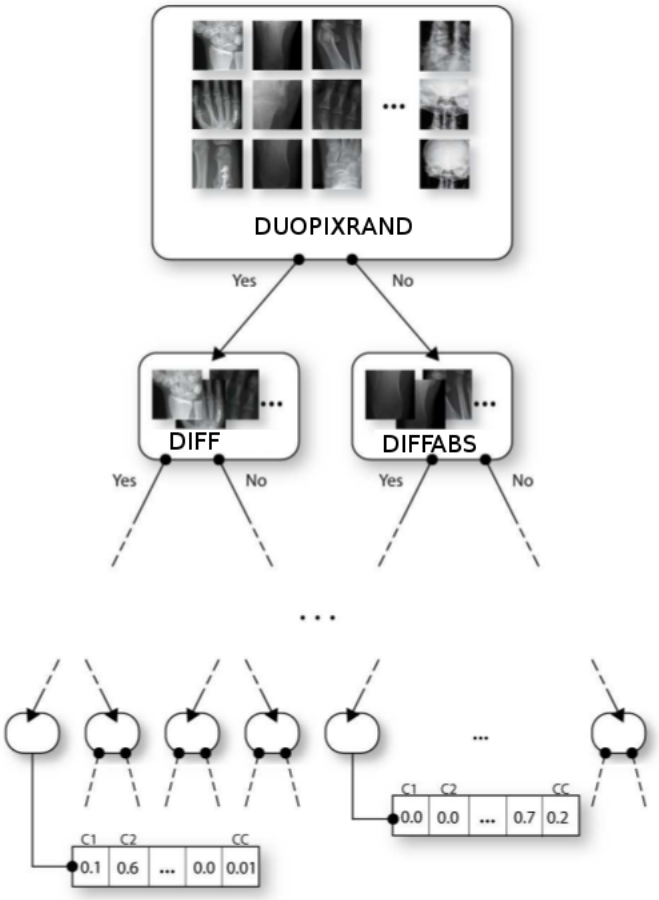


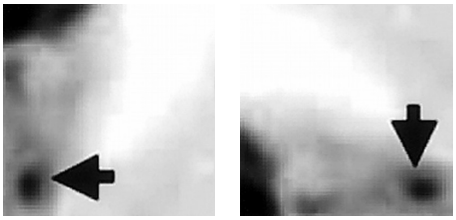
FIGURE 5.3: Exemple de construction d'arbre en mode RANDOM (adapté de [Mar09])

5.2 Au niveau des sous-fenêtres

Les variantes précédentes s'intéressaient aux changements influant directement la construction de l'arbre. Nous allons maintenant nous pencher sur les variantes qui agissent au moment de l'extraction de sous-fenêtres.

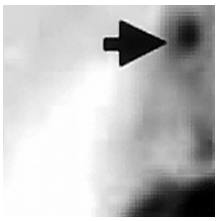
L'idée est ici d'appliquer une transformation géométrique aléatoire à la sous-fenêtre au moment de son extraction. Les transformations testées dans le cadre de ce travail sont de 2 types : rotations et flips.

Rotations Pour garder une sous-fenêtre carrée de même dimension dont les pixels ne sont pas altérés, seules 4 rotations sont envisagées : 0°, 90°, 180° et 270°. Ces rotations sont faites dans le sens trigonométrique (cf. figure 5.4).

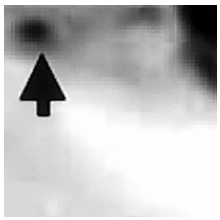


(a) 0°

(b) 90°



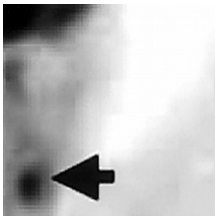
(c) 180°



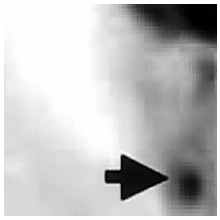
(d) 270°

FIGURE 5.4: Les 4 rotations possibles d'une sous-fenêtre

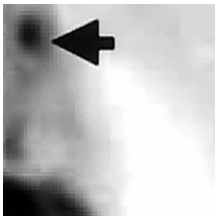
Flips Aux opérations de rotations nous ajoutons celles de flips. Un flip consiste à retourner l'image selon l'axe horizontal ou vertical passant par son centre. Il y a donc 3 flips possibles : aucun flip, un flip selon l'axe horizontal et un flip selon l'axe vertical (cf. figure 5.5).



(a) Aucun flip



(b) Flip selon l'axe vertical



(c) Flip selon l'axe horizontal

FIGURE 5.5: Les 3 flips possibles d'une sous-fenêtre

Donc, lorsque le programme est lancé en mode « rotate », chaque sous-fenêtre extraite se verra appliquer une rotation (parmi les 4 possibles) et un flip (parmi les 3 possibles) aléatoirement. Une même sous-fenêtre pourra donc être transformée de 12 façons différentes (cf. figure 5.6).

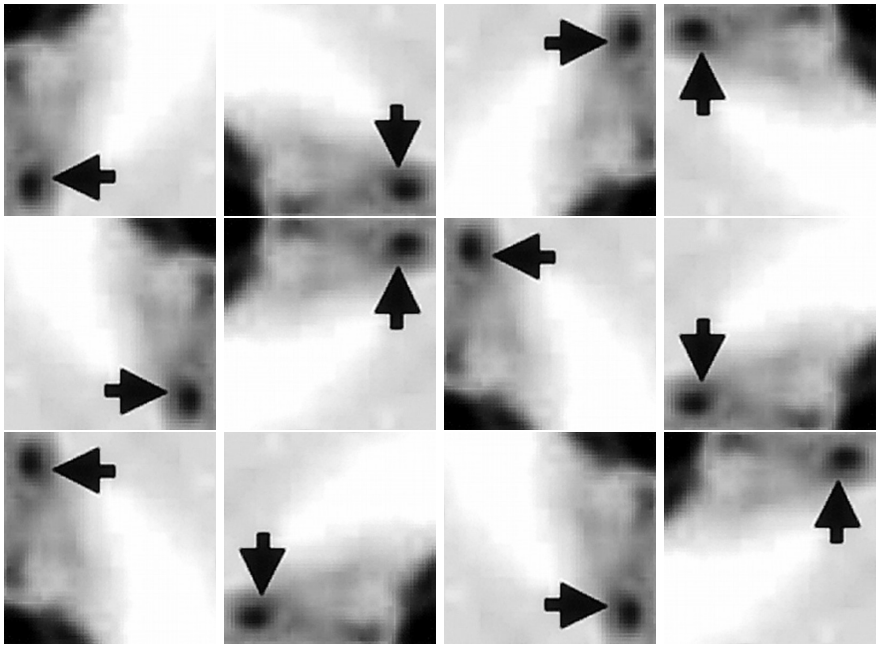


FIGURE 5.6: Les 12 orientations possibles d'une sous-fenêtre

Notons que les 4 rotations possibles suite à un flip horizontal (respectivement vertical) peuvent être obtenus par une autre rotation d'un flip vertical (respectivement horizontal). On le voit bien sur la figure 5.6, où la dernière ligne (rotations après flip selon l'axe horizontal) est composée des mêmes images que la deuxième ligne (rotations après flip selon l'axe vertical) mais dans un ordre différent puisque chaque colonne correspond à une même rotation.

5.3 Au niveau de l'image (filtres)

Une autre variante consiste à appliquer l'algorithme sur des images qui ont été préalablement filtrées. Nous allons considérer ici 2 types de filtres : les classiques et les aléatoires.

5.3.1 Filtres classiques

Les filtres que nous appelerons « classiques » regroupent un grand nombre de filtres souvent cités dans la littérature :
- Le gradient de Sobel
- Le gradient de Prewitt
- Le gradient de dérivée seconde
- Des filtres de détection de bords
- Des filtres médians

– Des filtres passe-haut, passe-bas, passe-bande
– Des filtres laplaciens
– etc.

Avant d'appliquer ces filtres à une image, celle-ci est redimensionnée à 10% de sa taille originale afin d'accélérer le calcul des filtres.

Au total, ce sont 190 filtres connus qui sont appliqués aux images, certains ne diffèrent que par la taille du masque de convolution appliqué (par exemple 3x3, 5x5 ou 9x9). Pour chaque image à filtrer, le script produit comme résultat une image sous format TIFF multipage contenant l'image originale ainsi que les 190 versions filtrées de cette image. Le programme gère ensuite ce format TIFF multipage si on lui indique qu'il doit le faire dans ses arguments (cf. section 6.1). Chaque sous-fenêtre extraite possède maintenant bien plus d'attributs, puisque comme l'illustre la figure 5.7 chaque fenêtre est décrite par toutes les images filtrées, ce qui fait au total 16x16x190=48640 attributs par sous-fenêtre.

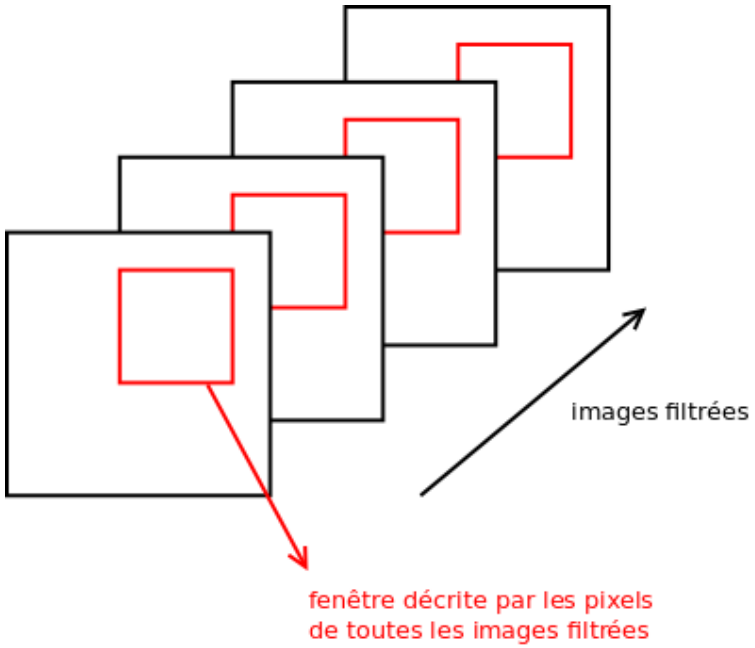


FIGURE 5.7: Extraction d'une sous-fenêtre d'un ensemble d'images filtrées

5.3.2 Filtres aléatoires

Nous opposons les filtres « classiques » aux filtres « aléatoires ». Ces derniers sont des filtres linéaires dont le masque de convolution a été généré aléatoirement. Dans le cadre de ce travail, nous utiliserons des masques de dimension 3x3 composés de coefficients entiers tiré aléatoirement selon une distribution uniforme dans l'intervalle [-10, 10].

Ces filtres s'inspirent en partie des réseaux à convolutions (cf. section 3.2.3). Nous avions d'abord tenté d'inclure ces convolutions aléatoires dans les tests des noeuds sous la forme de tests réalisant des combinaisons linéaires aléatoires d'attributs, mais cela entraîna des temps de calculs beaucoup trop longs, d'où l'idée d'effectuer préalablement les convolutions.

Le but de la manoeuvre n'est pas seulement de vérifier que les filtres ont un effet positif sur les résultats. Nous aimerions voir si c'est le fait d'utiliser des filtres « classiques » faisant ressortir certaines caractéristiques de l'image qui augmente les résultats, ou si appliquer n'importe quels filtres a le même effet sur le taux d'erreur de l'algorithme.

Chapitre 6

Implémentation

Nous allons aborder ici l'implémentation ainsi que la mise en pratique de l'algorithme.

6.1 PiXiT ©

PiXiT © est le programme sur lequel ce travail se base. Il est développé par la société PEPITe, qui a mis à la dispostion des chercheurs une version gratuite. Nous utilisons ici la dernière version du logiciel en ligne de commande fournie par Raphaël Marée, il existe une version avec interface graphique mais celle-ci est plus limitée dans ses options.

De plus, grâce à l'interface en ligne de commande nous pouvons lancer plusieurs instances du logiciel avec des paramètres différents. Toutes ces instances prenant beaucoup de temps à s'exécuter sur un ordinateur personnel, ce travail a pû bénéficier du supercalculateur (ou grille) du GIGA (Groupe Interdisciplinaire de Génoprotéomique Appliquée) de l'Université de Liège. Les différentes instances du programme pouvaient ainsi être exécutées sur cette grille grâce à un script shell, les résultats expérimentaux étant collectés dans les fichiers de sortie écrits par ces différentes instances.

6.2 Ajout des variantes

Passons maintenant à l'implémentation à proprement parler.

Types de tests aux noeuds

Les différents tests aux niveau des noeuds ont été programmés de façon à pouvoir facilement en ajouter d'autres par la suite. Nous sommes partis du code ExtraTrees.java du logiciel, que nous avons modifié pour l'ajout des différents tests.

Sans entrer dans les détails de l'implémentation du logiciel, ExtraTrees.java représente un ensemble d'arbres de décision et contient des méthodes permettant sa construction, sa sauvegarde et son chargement, sans oublier la propagation des images dans les arbres.

La méthode qui nous intéresse ici est *check_test*. Celle-ci prend en argument une sous-fenêtre, un attribut (pixel), un seuil et un index de noeud, et elle renvoie un booléen à vrai si la valeur correspondante au pixel est strictement inférieure au seuil, et faux sinon. Elle est utilisée à plusieurs endroits du code : lors de la construction d'un arbre et lors de la prédiction d'une nouvelle image. Dans la version de base, la valeur correspondante au pixel est toujours la valeur de ce pixel, mais dans nos variantes c'est justement sur cette valeur que nous allons jouer. C'est ici qu'intervient l'index de noeud : la méthode *check_test* doit savoir dans quel noeud elle se trouve pour pouvoir appliquer le bon test.

L'ensemble d'arbres généré doit donc maintenant posséder, en plus des attributs testés et des seuils de tests, le type de test appliqué à chaque noeud. Cela est réalisé en sauvegardant une liste de *NodeTest*. *NodeTest* est une interface contenant 2 types de fonctions (cf. diagramme 6.1) :
- *getTestVal* : étant donné une sous-fenêtre et un attribut (pixel) de celle-ci, cette fonction renvoie quelle valeur il faut comparer au seuil lors du test.
- *getExtraInfos, setExtraInfos* : accesseurs permettant de sauvegarder d'autres informations, comme l'autre pixel choisi au hasard dans un test DIFFRAND par exemple.

Chacun des tests décrits au chapitre 5 implémente donc cette interface, sauf RANDOM qui implique juste que le type de test d'un noeud est tiré aléatoirement parmi les autres types de tests.

Donc, lors du développement d'un noeud, ce n'est plus la valeur d'un pixel qui est systématiquement comparée à un seuil dans la méthode *check_test*, mais la valeur renvoyée par la méthode *getTestVal* du *NodeTest* du noeud actuel. Les bornes minimales et maximales dans lesquelles se trouvent le seuil de séparation sont aussi calculées selon les valeurs renvoyées par *getTestVal*.

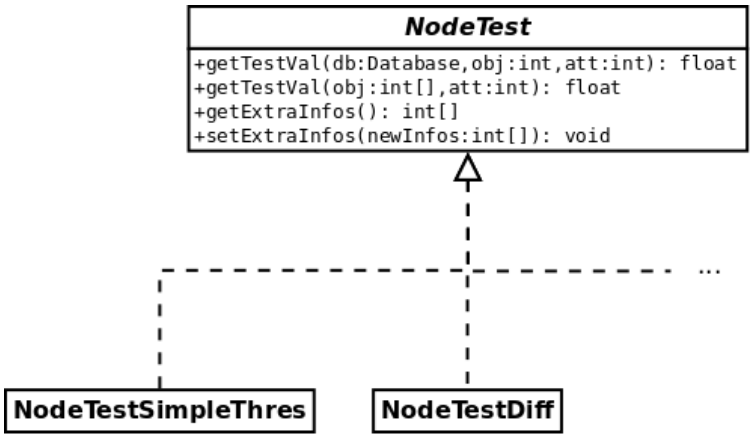


FIGURE 6.1: NodeTest

Un argument précisant le type de test au niveau des noeuds a été ajouté à la ligne de commande. Celui-ci est représenté par la chaîne de caractère de son nom comme présenté dans ce rapport

(SIMPLETHRES, DIFF, ...). Le programme récupère cette chaîne de caractères et l'envoie au singleton Protocol pour que cette information soit disponible partout dans le programme.

Rotations et flips

C'est au moment de l'extraction des sous-fenêtres que les rotations et flips aléatoires sont effectués. Cela est réalisé en concaténant plusieurs transformations affines (java.awt.geom.AffineTransform) tirées aléatoirement : on décide d'abord quelle rotation appliquée en choisissant un angle aléatoirement dans l'ensemble {0°, 90°, 180°, 270°}, puis on applique un flip nul, horizontal ou vertical, toujours choisi aléatoirement.

Filtres

Les filtres sont appliqués aux images avant de lancer PiXiT©, ils servent donc à créer un nouvel ensemble d'apprentissage. Ces filtres sont exécutés via des scripts faisant appel au programme *convert* de la suite ImageMagick®[1]. Les scripts pour les filtres classiques ont été fournis par Raphaël Marée, ceux pour les filtres aléatoires ont été écrits en adaptant ceux des filtres classiques. Les scripts prennent en entrée un dossier contenant des images sous format jpg ou png, et produisent en sortie dans ce même dossier des images TIFF multipages, chaque page représentant une version filtrée différente de l'image originale. Les images TIFF multipages sont lisibles par PiXiT©.

1. http ://www.imagemagick.org/

Chapitre 7

Bases de données

Cette section présente les différents ensembles d'apprentissage utilisés lors des tests. Comme le titre du travail le laisse entendre, les images contenues dans ces bases de données sont de nature biologiques.

7.1 IMAGECLEF2010

Cette base de données contient des images en couleurs et en niveaux de gris (pour l'apprentissage, pour le test). Elles sont extraites d'articles publiés dans des journaux scientifiques (« Radiology » et « Radiographics »). Elles ont été classées par des experts en 8 classes :
- CT : Tomographie.
- GX : Graphiques, principalement des dessins et graphes.
- MR : Imagerie à résonance magnétique.
- NM : Médecine nucléaire.
- PET : Tomographie à émission de positron.
- PX : Imagerie optique.
- US : Ultrason.
- XR : Rayons X.

FIGURE 7.1: Exemples d'images appartenant à IMAGECLEF2010

Le protocole de test pour cette base est le suivant : test indépendant (les images pour la construction du modèle et pour les tests sont différentes), 2390 images pour l'apprentissage, 2620 pour les tests. La taille des sous-fenêtres varie de 0 à 10% de la taille de l'image originale.

7.2 SUBCELLULAR

Cette base de données contient 862 images de taille 512 × 382 pixels, en niveaux de gris. Les images sont séparées en 10 classes : ActinFilaments, Endosome, ER, Golgi gia, Golgi gpp, Lysosome, Microtubules, Mitochondria, Nucleolus, et Nucleus. Ces images ont été acquises par le Murphy Lab [Mur].

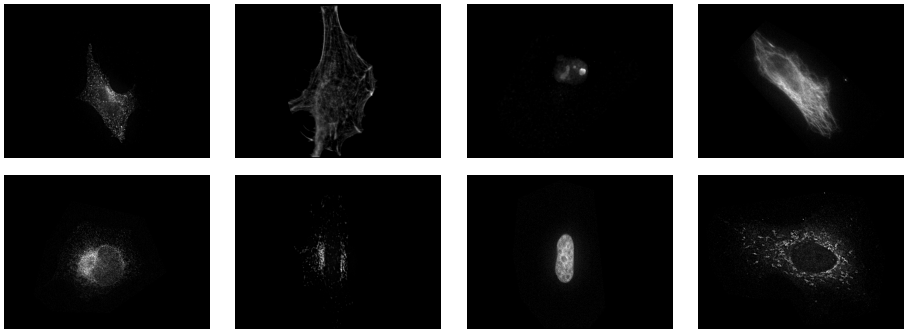


FIGURE 7.2: Exemples d'images appartenant à SUBCELLULAR

Le protocole de test pour cette base est le suivant : échantillonner 10 fois en prenant 90% de la totalité des images par classe pour l'apprentissage, et les 10% restant pour le test. La taille des sous-fenêtres varie de 50 à 100% de la taille de l'image originale.

7.3 RBC

RBC (Red Blood Cell) est composée d'images de globules rouges. On y trouve 3 classes, qui correspondent à différentes formes de globules rouges (stomatocyte, discocyte et echinocyte). Cette base de données contient un total de 5062 images en niveaux de gris.

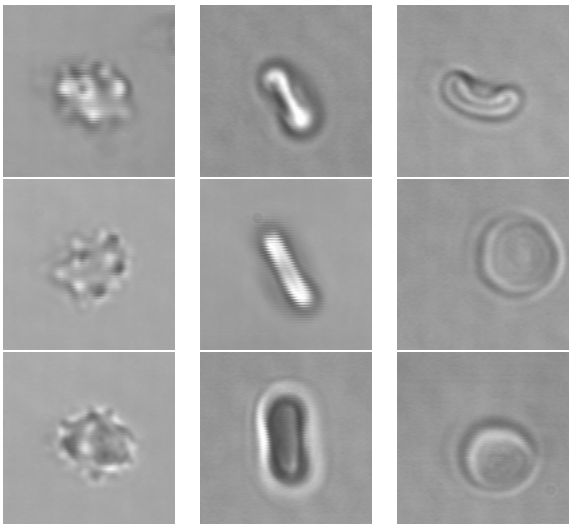


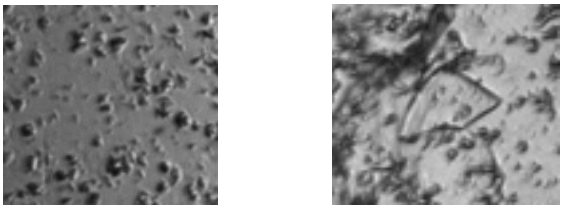
FIGURE 7.3: Exemples d'images appartenant à RBC

Le protocole de test pour cette base est le suivant : échantillonner 10 fois 1500 images pour l'apprentissage (500 par classe), en prenant le reste des images pour les tests. La taille des sous-fenêtres

varie de 50 à 75% de la taille de l'image originale.

7.4 CRYSTAL

Ce sont des images de crystallographie. Elles représentent des régions contenant ou non un crystal. Elles proviennent de l'Université d'Aachen (RWTH), plus précisément du Fraunhofer-Institut für Molekularbiologie und Angewandte Oekologie IME où travaille le Prof. K. Hoffmann. Cette base de données contient un total de 3907 images en niveaux de gris.



(a) Pas de crystal (b) Crystal

FIGURE 7.4: Exemples d'images appartenant à CRYSTAL

Le protocole de test pour cette base est le suivant : échantillonner 10 fois en prenant 80% de la totalité des images par classe pour l'apprentissage, et les 20% restant pour le test. La taille des sous-fenêtres varie de 0 à 100% de la taille de l'image originale.

7.5 IRMA2005

IRMA2005 (Image Retrieval in Medical Applications) est un ensemble contenant 10000 images Rayon-X qui ont été prises par le département de radiologie diagnostique de l'Aachen University of Technology (Allemagne) en 2005. L'ensemble d'apprentissage contient 9000 images et est divisé en 57 classes ; l'ensemble de test contient 1000 images. Ces photos ont une dimension maximum de 512 x 512 pixels et sont en niveaux de gris. Elles couvrent différents âges, genres, angles de prise de vue et pathologies. La qualité des images peut donc varier significativement.

FIGURE 7.5: Exemples d'images de la base IRMA

Le protocole de test pour cette base est un test indépendant : 9000 images pour l'apprentissage, 1000 pour les tests. La taille des sous-fenêtres varie de 75 à 100% de la taille de l'image originale.

Chapitre 8

Résultats

Nous allons maintenant commenter les résultats obtenus par les différentes variantes présentées au chapitre 5 sur les bases de données énoncées au chapitre 7. Afin de rendre la lecture de ce chapitre plus agréable, tous les graphiques (à l'exception des récapitulatifs) ont été placés à la fin de ce document (cf. annexe A).

8.1 Au niveau des noeuds et de l'orientation des sous-fenêtres

8.1.1 Classification Directe

Commençons par les résultats de l'algorithme de classification directe décrit à la section 4.1. Pour chaque ensemble d'apprentissage nous lançons plusieurs fois le programme en faisant varier le nombre de sous-fenêtres ou le nombre d'arbres, et nous observons l'effet de ces variations sur l'erreur de classification moyenne.

Le paramètre k (nombre de tests au sein des noeuds des arbres) utilisé lors de ces tests sur cette base est de 16, sauf pour IMAGECLEF2010 où k est à 28.

IMAGECLEF2010

Comme on peut le voir sur la figure A.1, augmenter le nombre de sous-fenêtres diminue l'erreur moyenne pour la plupart des tests comme déjà observé dans [MGPW05]. Mis à part les tests DIFF, DIFFABSRAND, DIFFTWOPIXLBP et DIFFABS qui sont moins bons, tous les autres tests ont des performances équivalentes à l'algorithme de base (à savoir SIMPLETHRES).

Observons maintenant l'effet d'une augmentation du nombre d'arbres sur la figure A.2. Les tests DIFF, DIFFABSRAND, DIFFTWOPIXLBP et DIFFABS ont toujours des résultats médiocres par rapport aux autres tests. On remarque aussi que les performances des autres tests collent toujours à celle de SIMPLETHRES, seul DIFFRAND se distingue très légèrement sur 20 et 50 arbres.

Rotations et flips En faisant subir des rotations et flips aléatoires aux sous-fenêtres, on ne remarque pas de gros changements aux niveaux des résultats (cf. figure A.3 et A.4). On décèle une erreur légèrement plus haute, qui fait que les résultats ont un peu plus de mal à passer sous la barre des 45% d'erreur moyenne lorsqu'on augmente les paramètres.

Couleurs Comme la base IMAGECLEF2010 contient des images en couleurs, il est intéressant de remarquer que tenir compte des couleurs améliore clairement les résultats de l'algorithme (cf. figures A.5, A.6, A.7 et A.8).

Ici nous avons comparé à titre d'exemple les améliorations sur les tests SIMPLETHRES et DIFFABS. Ce sont maintenant les composantes H (Hue (*Teinte*) dans l'espace HSV) qui sont soustraite dans le test DIFFABS, et plus le niveaux de gris. La plage de valeurs de H ([0, 360]) étant plus étendue que celle du niveau de gris ([0, 255]), il parait normal d'observer une amélioration du classement des images en couleurs par rapport à celles en niveaux de gris.

On remarque que globalement SIMPLETHRES gagne un peu plus que son homologue au passage en couleur.

SUBCELLULAR

Dans cette base de données, on remarque tout de suite que les tests DIFF, DIFFTWOPIXLBP et DIFFABS performent moins bien que les autres, peu importe qu'on augmente le nombre de sous-fenêtres ou le nombre d'arbres (cf. figures A.15 et A.16). On remarque que le test RANDOM se distingue des autres par une erreur moyenne plus faible lorsqu'on augmente les paramètres.

Rotations et flips Les rotations et flips aléatoires ne changent pas la donne (cf. figures A.17 et A.18), les tests perfoment globalement moins bien. DUOPIX a maintenant le meilleur score, mais seulement lorsqu'on prend un grand nombre de sous-fenêtres (200, pour 10 arbres). C'est SIMPLETHRES le meilleur lorsque le nombre d'arbres est élevé (50, avec 50 sous-fenêtres).

RBC

Ici, on remarque une nette différence entre les test DIFF et DIFFTWOPIXLBP et les autres (cf. figures A.25 et A.26), ceux-ci ont une erreur moyenne bien plus faible que le test standard SIMPLETHRES. Les résultats du test DIFFABS rejoignent ceux de DIFF lorsque le nombre de sous-fenêtres ou le nombre d'arbres est élevé.

Rotations et flips Les rotations et flips aléatoires ont ici un effet positif puisqu'ils font baisser les erreurs moyennes de tous les tests (cf. figures A.27 et A.28). Ici l'allure des graphes a quelque peu changée : la courbe du test DIFFABS épouse maintenant celle de DIFF. DIFF est maintenant le meilleur test, juste devant DIFFTWOPIXLBP et DIFFABS.

CRYSTAL

On constate ici qu'augmenter le nombre de sous-fenêtres a peu d'influence sur l'erreur moyenne (cf. figure A.35). Lorsqu'on a 50 sous-fenêtres ou plus, seuls les tests DIFF et DIFFABS continuent de baisser significativement et obtiennent une erreur moyenne plus faible que les autres tests.

L'effet de l'augmentation du nombre d'arbres est quelque peu différent suivant les tests (cf. figure A.36). Augmenter le nombre d'arbres diminue l'erreur moyenne des tests SIMPLETHRES, LOGSIMPLEPIX et DIFFABS, en revanche les autres tests voient leur erreur augmenter à partir de 20 arbres (ou même 10 pour le test DIFF). On remarque aussi facilement que le test DIFFABS se distingue en ayant une erreur moyenne plus faible que les autres tests à partir de 20 arbres.

Rotations et flips Les rotations et flips aléatoires n'améliorent pas les résultats sur cette base de données (cf. figures A.37 et A.38). Le test DIFFABS reste le meilleur mais obtient une erreur moyenne un peu supérieure à celle qu'il avait sans rotations et flips aléatoires.

IRMA

Ici, comme on peut le voir sur les figures A.45 et A.46, seul le test DUOPIX a un taux d'erreur légèrement inférieur au test SIMPLETHRES. Par contre les tests DIFFABS, DIFFABSRAND, DIFFTWOPIXLBP et DIFF ont un taux d'erreur bien plus important.

Rotations et flips Avec les rotations et flips aléatoires (figures A.47 et A.48) on note que les tests DUOPIX et DIFFRAND sont les plus proche de SIMPLETHRES. DIFFABS, DIFFABSRAND, DIFFTWOPIXLBP et DIFF ont toujours un taux d'erreur plus important.

Récapitulatif

Les résultats précédents amenant beaucoup d'informations à la fois, essayons de répondre à la question qui nous intéresse : a-t-on fait mieux que SIMPLETHRES ?

Comme on peut le voir sur la figure 8.1, nous avons chaque fois un des autres tests qui a un taux d'erreur plus bas que SIMPLETHRES, même si la différence n'est pas toujours très grande.

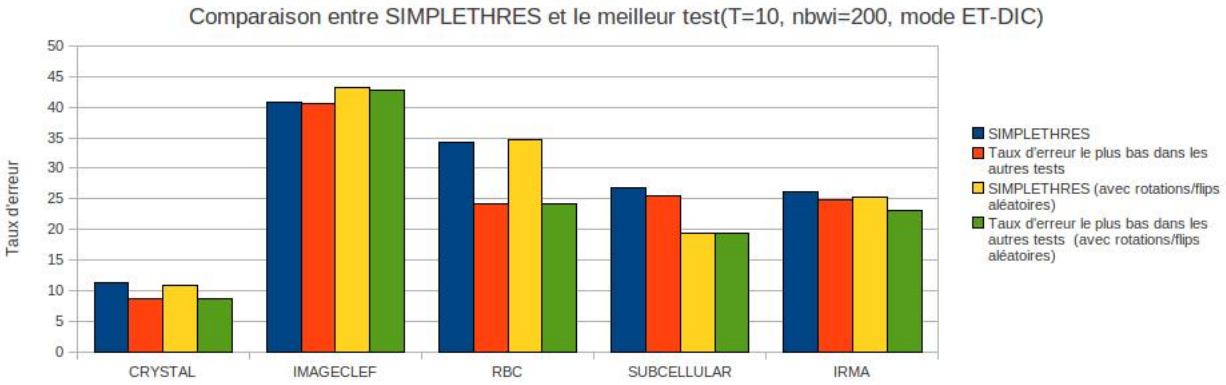


FIGURE 8.1: Récapitulatif des tests en classification directe sur les différentes bases de données, avec et sans rotations/flips aléatoires

Y a-t-il un test en particulier qui se dégage des autres ? Hélas non, le tableau 8.1 nous présente les meilleurs tests par base de donnée, et on remarque tout de suite la diversité de ceux-ci.

	Sans rotations et flips aléatoires	Avec rotations et flips aléatoires
CRYSTAL	DIFF	DIFFABS
IMAGECLEF	DUOPIX	LOGSIMPLEPIX
RBC	DIFFTWOPIXLBP	DIFF
SUBCELLULAR	RANDOM	DUOPIX
IRMA	DUOPIX	DIFFRAND

TABLE 8.1: Récapitulatif des meilleurs tests en classification directe

8.1.2 Bags of Features

Passons maintenant aux résultats pour le mode BAGS (cf. section 4.2). Le paramètre k (nombre de tests au sein des noeuds des arbres) utilisé lors de ces tests sur cette base est de 16, sauf pour IMAGECLEF2010 où k est à 28.

IMAGECLEF2010

Pour cette base, aucun test n'arrive à battre SIMPLETHRES (cf. figures A.9 et A.10). On remarque que les tests se basant sur un autre pixel choisi aléatoirement (DIFFABSRAND, DUOPIXRAND, DIFFRAND et DIFFTWOPIXLBP) ont un taux d'erreur nettement supérieur aux autres.

SUBCELLULAR

Ici non plus aucun test ne sort du lot (figures A.19 et A.20). SIMPLETHRES est légèrement battu par LOGSIMPLEPIX sans les rotations et flips aléatoires.

Avec les rotations et flips aléatoires, le test SIMPLETRHES a clairement un taux d'erreur inférieur aux autres tests.

RBC

On remarque sur les figures A.29 et A.30 plusieurs tests faisant mieux que SIMPLETHRES. Celui ayant le meilleur score, avec ou sans rotations et flips aléatoires, est DIFF.

CRYSTAL

Pour la base CRYSTAL on remarque encore que quelques tests arrivent à battre SIMPLETHRES (cf. figures A.39 et A.40). Celui sortant largement du lot étant RANDOM.

IRMA

On remarque que sur la base IRMA les tests faisant intervenir un deuxième pixel aléatoire on un taux d'erreur bien plus élevé que les autres tests (cf. figures A.49 et A.50), comme nous l'avions remarqué sur la base IMAGECLEF2010. DIFF est le test se comportant le mieux, avec ou sans les rotations et flips aléatoires des sous-fenêtres.

Récapitulatif

Tout comme en classification directe, on a ici aussi souvent un test obtenant un meilleur taux d'erreur que SIMPLETHRES, comme on peut le voir sur le graphique 8.2.

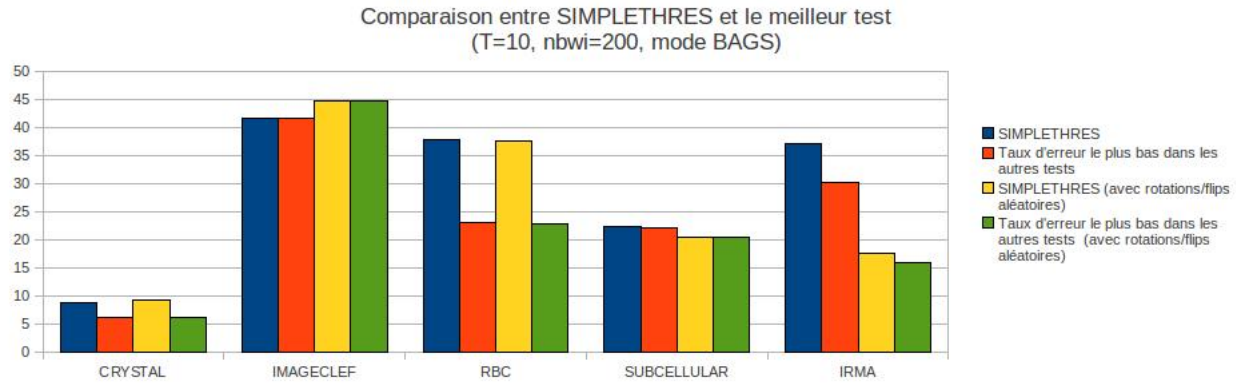


FIGURE 8.2: Récapitulatif des tests en mode BAGS sur les différentes bases de données, avec et sans rotations/flips aléatoires

On ne remarque pas ici non plus de test se distinguant particulièrement, même si RANDOM et DIFF reviennent assez souvent comme on peut le voir sur le tableau 8.2.

	Sans rotations et flips aléatoires	Avec rotations et flips aléatoires
CRYSTAL	RANDOM	RANDOM
IMAGECLEF	SIMPLETHRES	RANDOM
RBC	DIFF	DIFF
SUBCELLULAR	LOGSIMPLEPIX	SIMPLETHRES
IRMA	DIFF	DIFF

TABLE 8.2: Récapitulatif des meilleurs tests en mode BAGS

8.1.3 Variation de k

Intuitivement, nous pourrions penser que augmenter k, le nombre de tests évalués en chaque noeud, diminuerait fortement l'erreur de la plupart des tests, et en particulier du test RANDOM puisque celui-ci peut à la fois varier sur les attributs choisis et sur la nature du test. Si cette intuition se confirme, cela nous permettrait de dire que RANDOM est un bon test universel, donnant de bons résultats (meilleurs que SIMPLETHRES) sur toutes les bases de données testées.

Voyons ce qu'il en est en comparant les erreurs de SIMPLETHRES et RANDOM pour des valeurs de k de 1, 16 (ou 28 pour IMAGECLEF2010) et 256. Ces tests ont été effectués en mode BAGS, avec 10 arbres et 200 sous-fenêtres extraites par image.

IMAGECLEF2010

Sur IMAGECLEF2010 avec ou sans rotations et flips aléatoires (figures A.11 et A.12), RANDOM fait aussi bien que SIMPLETHRES, mais pas mieux, même pour k=256.

SUBCELLULAR

Ici non plus RANDOM ne se distingue pas (cf. figures A.21 et A.22), même s'il fait un peu mieux que SIMPLETHRES avec les rotations et flips aléatoires.

RBC

Nous avons ici représenté en plus le test DIFF car il donnait des résultats nettement meilleurs que SIMPLETHRES et RANDOM en mode BAGS. On voit sur les figures A.31 et A.32 qu'il reste le meilleur en augmentant k.

CRYSTAL

Pas de grosses différences ici (cf. figures A.41 et A.42), RANDOM reste meilleur que SIMPLETHRES, mais augmenter k n'a pas de gros impacts sur le taux d'erreur.

IRMA

Tout comme pour CRYSTAL, augmenter k ne change presque rien lorqu'on n'effectue pas de rotations et flips aléatoires, et RANDOM reste un peu meilleur que SIMPLETHRES (cf. figures A.51 et A.52).

Récapitulatif

Bien qu'on remarque souvent une baisse de l'erreur du test RANDOM, celle-ci n'est pas spectaculaire. Nous avons néanmoins vu que RANDOM fait toujours au moins aussi bien que SIMPLETHRES lors de ces tests.

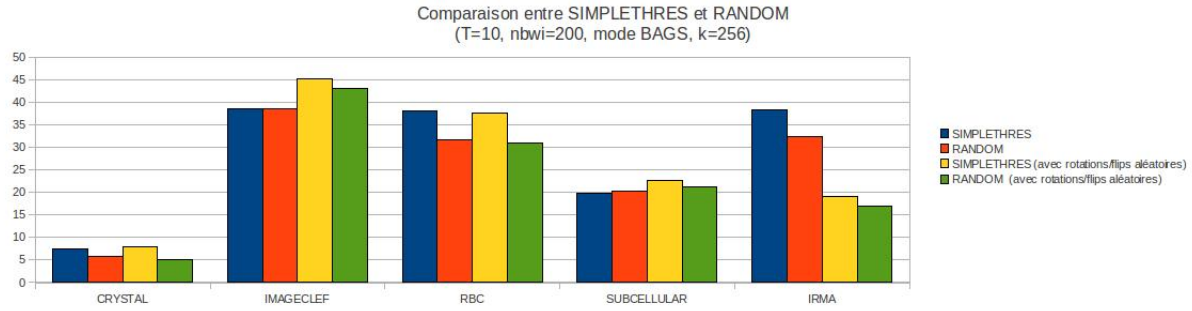


FIGURE 8.3: Comparaison entre SIMPLETHRES et RANDOM pour k=256

8.2 Filtres

Comparons maintenant les résultats des différents tests sur les versions filtrées des images. Le paramètre k (nombre de tests au sein des noeuds des arbres) utilisé lors de ces tests sur cette base est de 16, sauf pour IMAGECLEF2010 où k est à 28.

IMAGECLEF2010

On voit que sur les figures A.13 et A.14 qu'appliquer les filtres classiques diminue l'erreur quelque soit le test utilisé, par contre les convolutions aléatoires se comportent souvent moins bien.

SUBCELLULAR

On voit sur les figures A.23 et A.24 que les filtres classiques obtiennent sur la plupart des tests des résultats légèrement meilleurs que SIMPLETHRES. Encore une fois les convolutions aléatoires ont des taux d'erreur presque toujours supérieurs à ceux de SIMPLETHRES.

RBC

Sur RBC, les filtres classiques et les convolutions aléatoires se comportent mal et obtiennent toujours des taux d'erreurs largement supérieurs à ceux de SIMPLETHRES (cf. figures A.33 et A.34).

CRYSTAL

Sur la base CRYSTAL, les résultats sont plus mitigés. On voit sur les figures A.43 et A.44 que les convolutions aléatoires ont souvent un taux d'erreur inférieur aux versions sans filtres ou avec filtres classiques.

IRMA

La base IRMA a des résultats très différents suivant qu'on travaille avec ou sans les rotations et flips aléatoires. Sans rotations (figure A.53), les convolutions aléatoires obtiennent toujours des taux d'erreur inférieurs, quel que soit le test. Par contre avec les rotations et flips aléatoires (figure A.54), les versions filtrées (classiques ou convolutions aléatoires) ont toujours des taux d'erreurs supérieurs aux versions sans filtres, sauf sur les tests prenant en compte un deuxième pixel aléatoire (DIFFABSRAND, DUOPIXRAND, DIFFRAND et DIFFTWOPIXLBP).

Récapitulatif

Au final, nous ne pouvons pas dire qu'appliquer des filtres diminue toujours le taux d'erreur, cela est fort dépendant de la base de donnée.

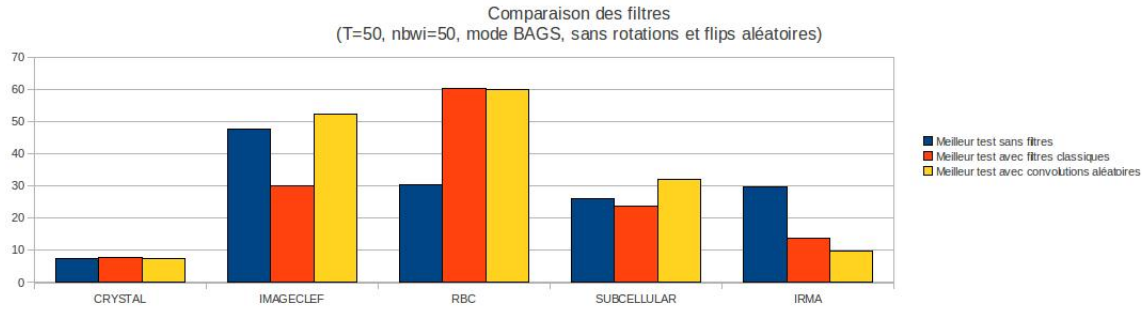


FIGURE 8.4: Comparaisons des résultats avec et sans filtres

Chapitre 9

Conclusion

L'objectif de ce travail était de proposer, d'implémenter et d'étudier les performances de plusieurs variantes basées sur les arbres de décisions.

Après avoir rappelé quelques concepts d'apprentissage automatique et avoir passé en revue quelques algorithmes populaires en classification d'images, nous avons détaillé l'algorithme sur lequel se base les variantes de ce travail : les Extra-Trees à sous-fenêtres aléatoires.

Ensuite, les différentes variantes ont été présentées dans le chapitre 5. Celles-ci peuvent être divisées en 3 catégories :
- les variantes de tests au niveau des noeuds des arbres, modifiant la façon dont les arbres sont construits ;
- les changements d'orientations des sous-fenêtres :
  - rotations aléatoires ;
  - flips aléatoires ;
- les filtres :
  - les filtres dits « classiques », ceux que l'ont retrouvent le plus souvent dans la littérature (Sobel, Prewitt, médians, ...) ;
  - les convolutions aléatoires.

Les taux d'erreurs de ces variantes ont été étudiés dans le chapitre 8 sur 5 bases de données contenant des images de nature biologique : CRYSTAL, IMAGECLEF2010, IRMA2005, SUBCELLULAR et RBC.

Plusieurs variantes ont donné de meilleurs résultats que l'algorithme de base. Mais les variantes de tests au niveau des noeuds des arbres donnant les meilleurs résultats ne sont pas les mêmes suivant la base de données. Nous avons donc implémenté un test RANDOM, qui choisit aléatoirement quel test appliquer pour un noeud donné. Comme le nombre de tests évalué en chaque noeud est fixé par le paramètre k, nous avons ensuite étudié l'effet de l'augmentation de k sur l'algorithme de base et sur les tests RANDOM. Nous avons remarqué que RANDOM obtenait des résultats au moins aussi bons que ceux de l'algorithme de base, quelle que soit la base de données.

Nous avons ensuite étudié l'effet des filtres sur les résultats. Celui-ci est fort variable suivant la base de données.

D'autres variantes pourraient être envisagées lors de futures recherches. Il serait effectivement intéressant d'étudier les performances de tests au niveau des noeuds qui prendraient en compte plusieurs pixels d'une image (comme le font déjà certains tests présentés dans ce travail) mais sur des versions filtrées différentes de cette image. On pourrait ainsi prendre en compte des caractéristiques mises en évidence par différents filtres. Par exemple, si l'on a un filtre détectant les bords horizontaux et un autre détectant les bords verticaux, de tels tests pourraient prendre en compte les coins.

Tous ces tests ont pris un certain temps à être réalisés car ils consomment beaucoup de ressources matérielles. Mais comme rien n'arrête le progrès, nous pensons que les puissances de calcul et les capacités de stockage vont continuer à augmenter, pour des prix toujours plus compétitifs, ce qui permettra aux algorithmes d'apprentissage d'être de plus en plus répandus.

Annexe A

Graphiques des résultats

A.1 IMAGECLEF2010

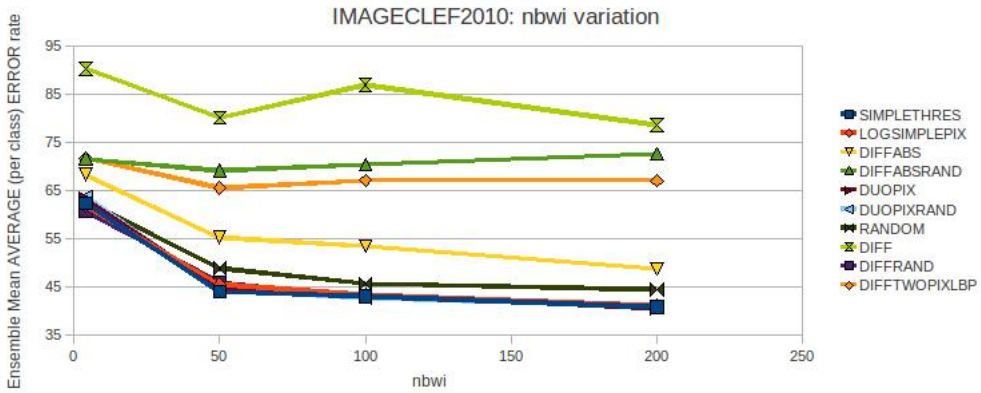


FIGURE A.1: IMAGECLEF2010 : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres en classification directe

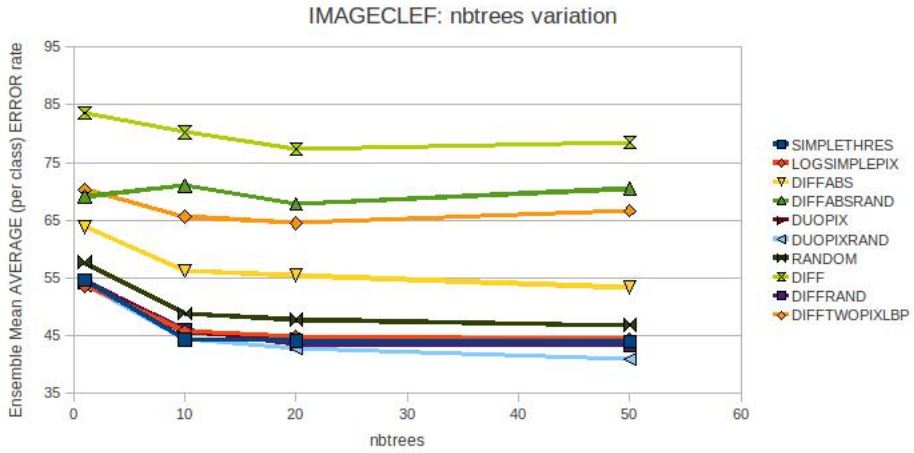


FIGURE A.2: IMAGECLEF2010 : Erreur moyenne sur les tests simples avec variation du nombre d'arbres en classification directe

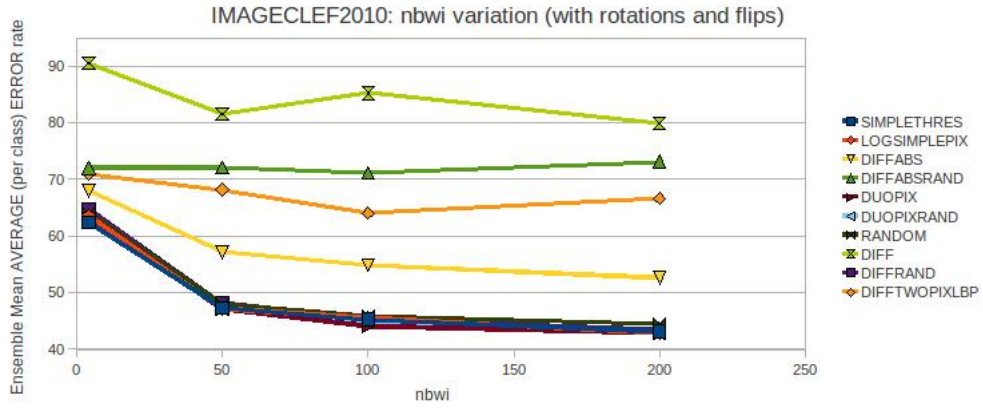


FIGURE A.3: IMAGECLEF2010 : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

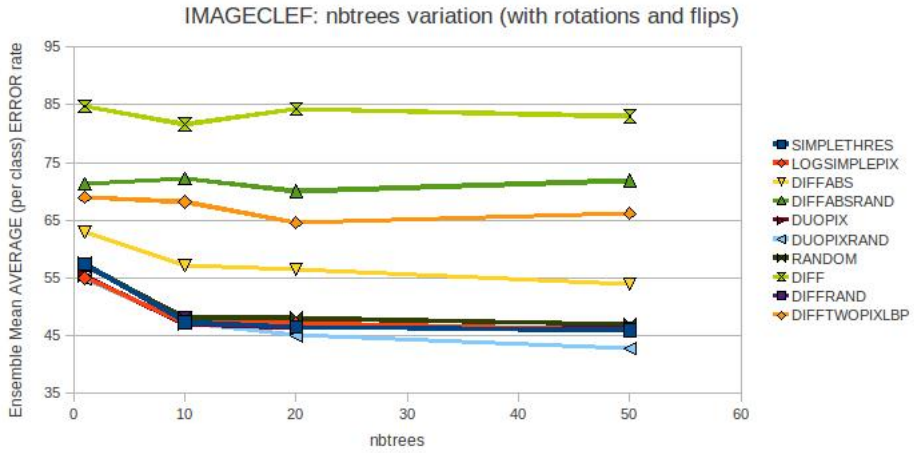


FIGURE A.4: IMAGECLEF2010 : Erreur moyenne sur les tests simples avec variation du nombre d'arbres et rotations/flips aléatoires en classification directe

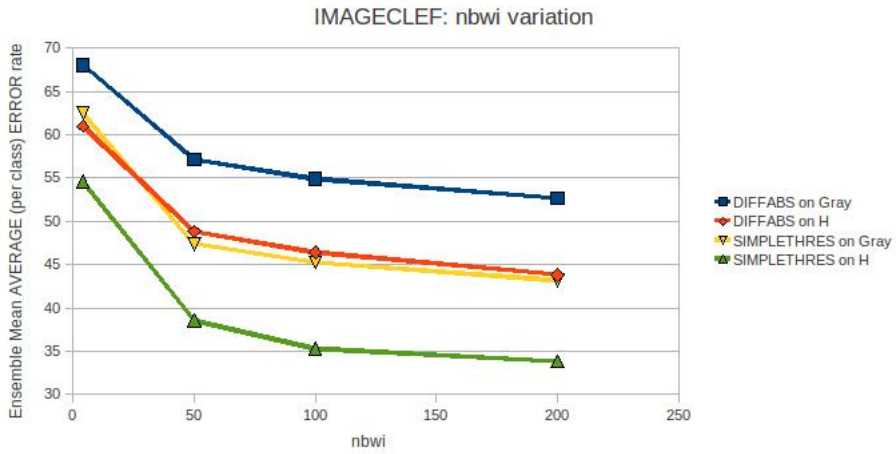


FIGURE A.5: IMAGECLEF2010 : Comparaison gris/couleurs avec variation du nombre de sous-fenêtres en classification directe

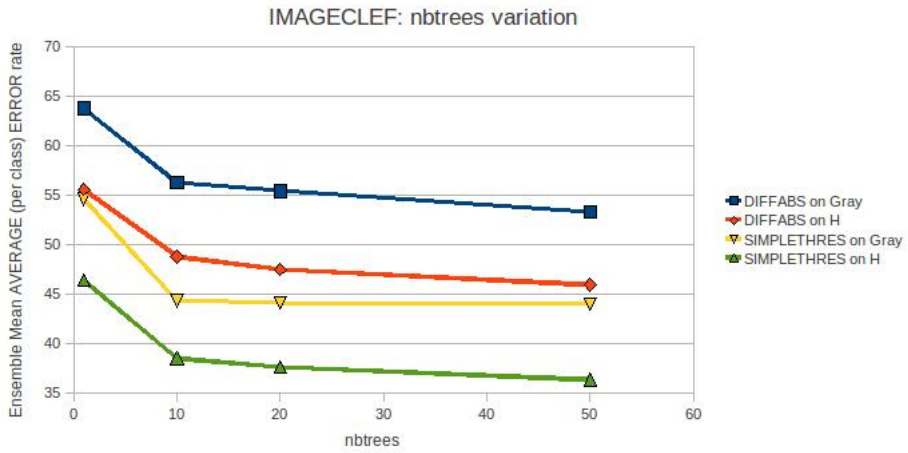


FIGURE A.6: IMAGECLEF2010 : Comparaison gris/couleurs avec variation du nombre d'arbres en classification directe

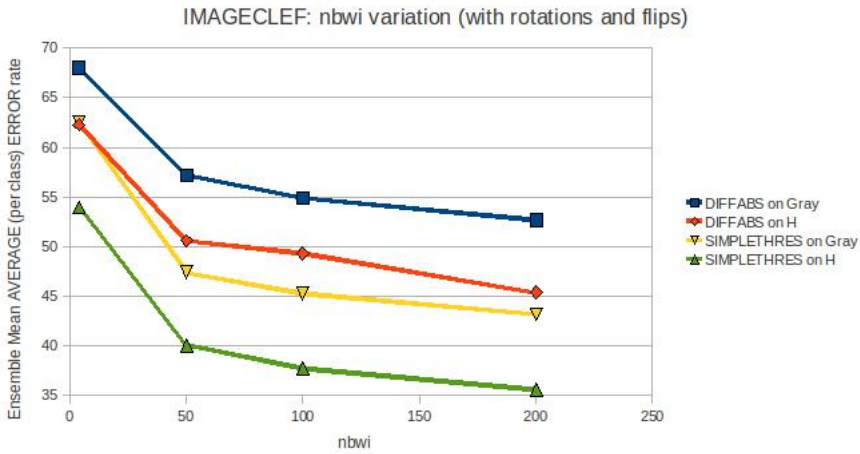


FIGURE A.7: IMAGECLEF2010 : Comparaison gris/couleurs avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

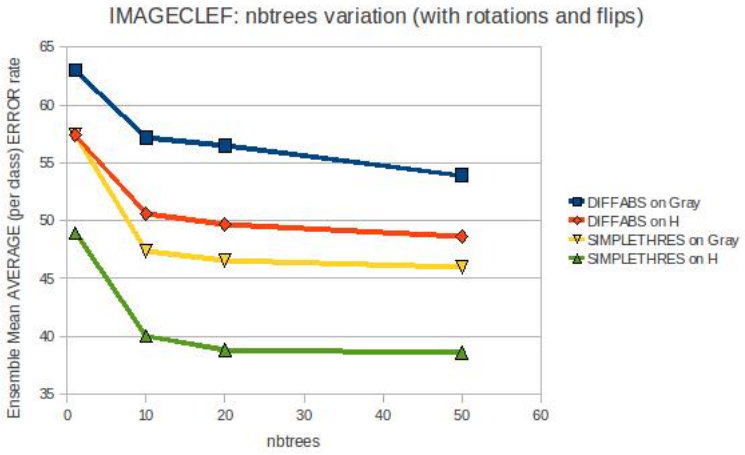


FIGURE A.8: IMAGECLEF2010 : Comparaison gris/couleurs avec variation du nombre d'arbres et rotations/flips en classification directe

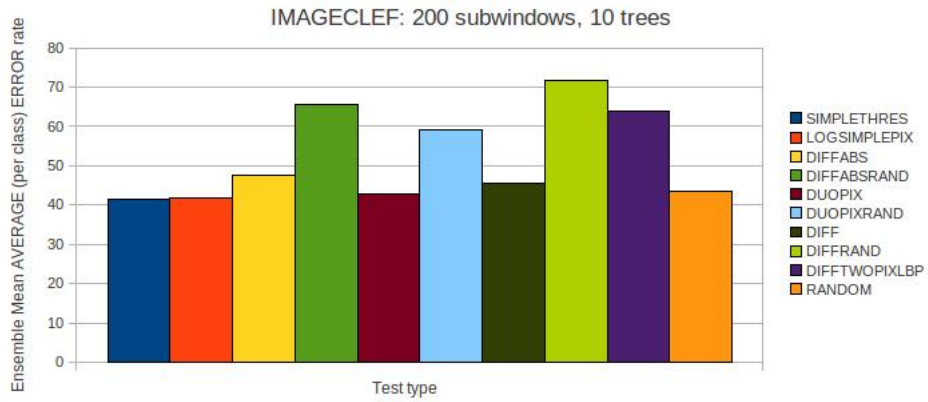


FIGURE A.9: IMAGECLEF : Erreur moyenne sur les différents tests en mode BAGS

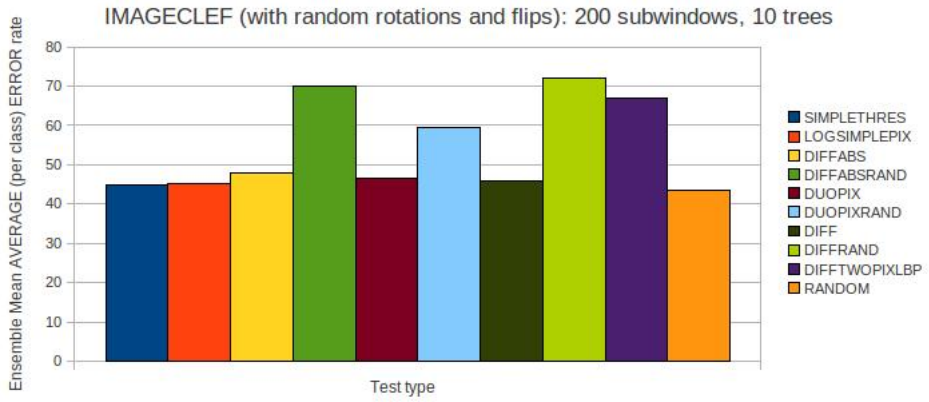


FIGURE A.10: IMAGECLEF : Erreur moyennesur les différents tests en mode BAGS avec rotations et flips aléatoires

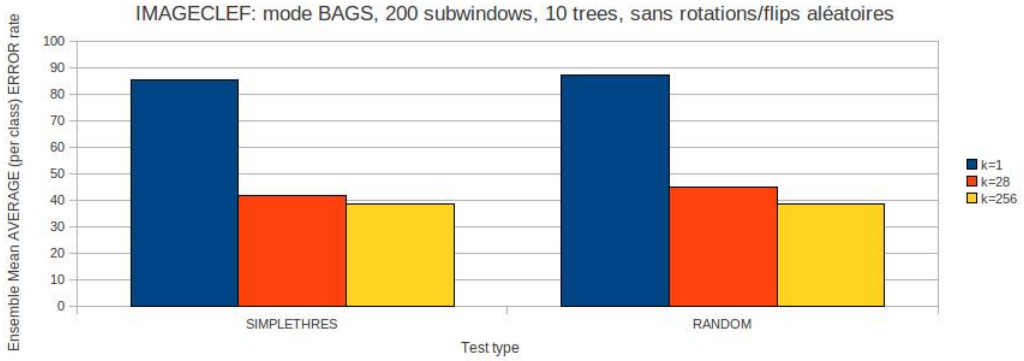


FIGURE A.11: IMAGECLEF2010 : Variation de k, sans rotations et flips

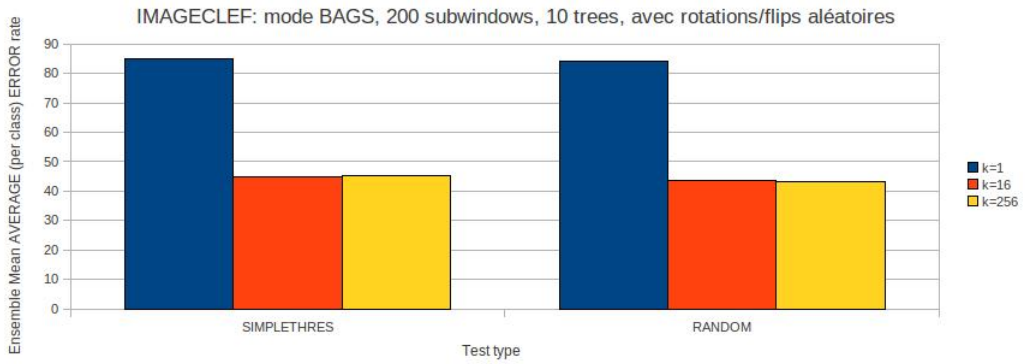


FIGURE A.12: IMAGECLEF2010 : Variation de k, avec rotations et flips

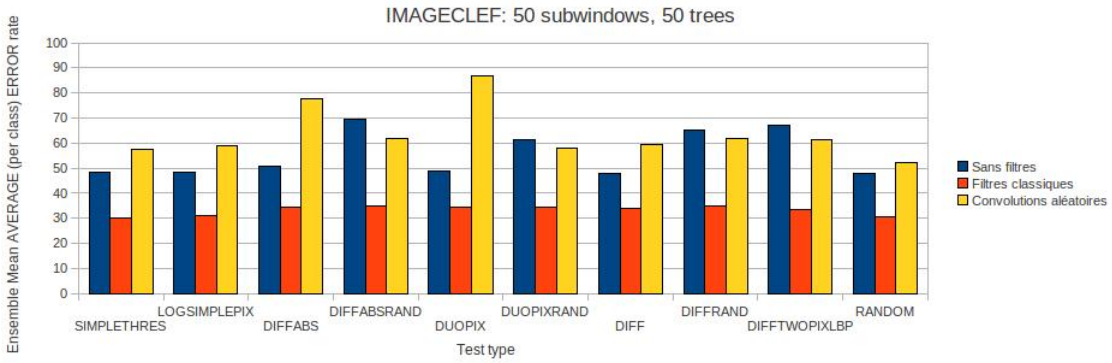


FIGURE A.13: IMAGECLEF2010 : Filtres sans rotations et flips

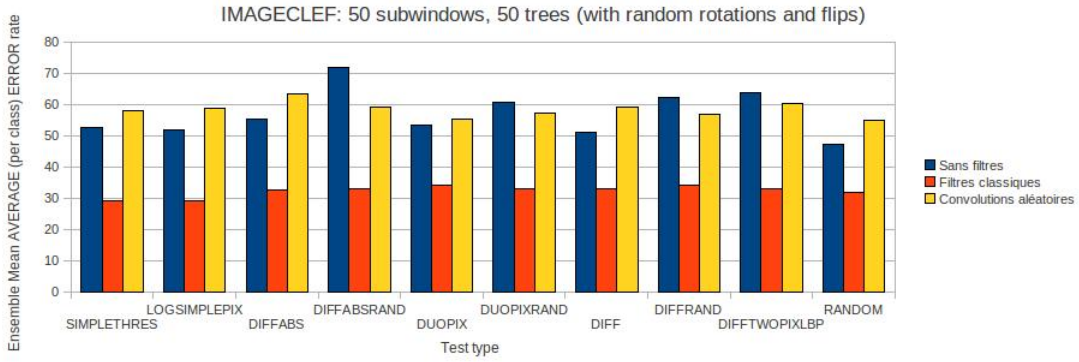


FIGURE A.14: IMAGECLEF2010 : Filtres avec rotations et flips

A.2 SUBCELLULAR

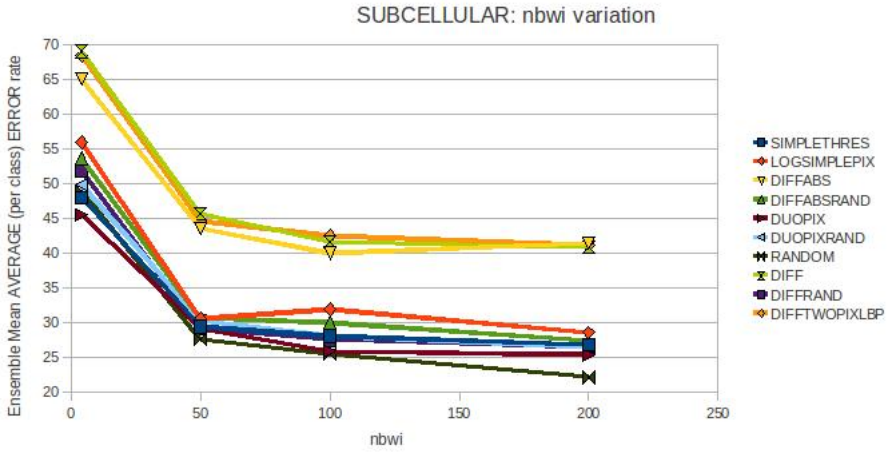


FIGURE A.15: SUBCELLULAR : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres en classification directe

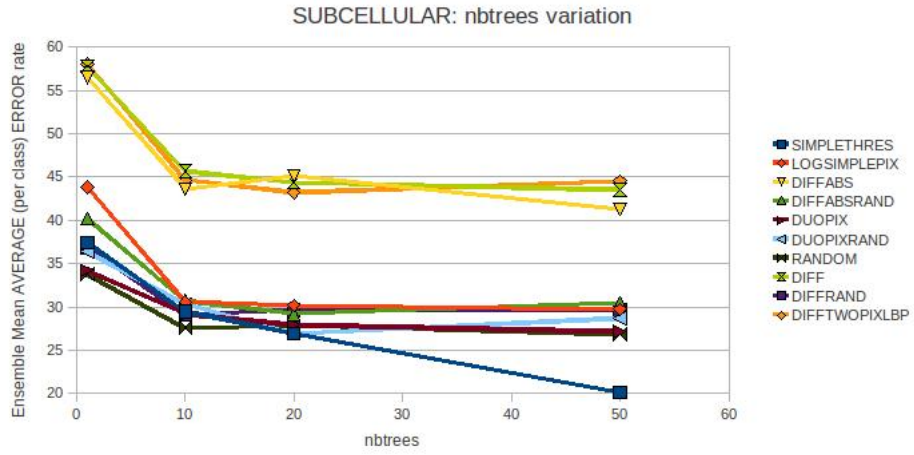


FIGURE A.16: SUBCELLULAR : Erreur moyenne sur les tests simples avec variation du nombre d'arbres en classification directe

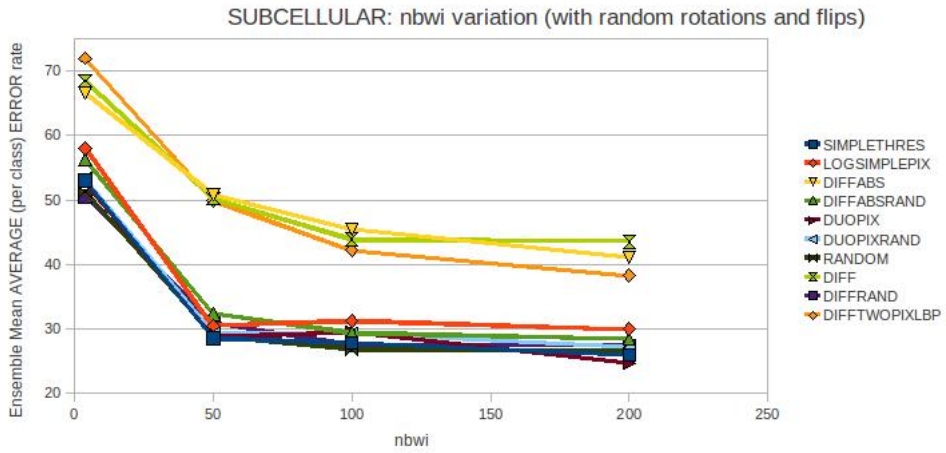


FIGURE A.17: SUBCELLULAR : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

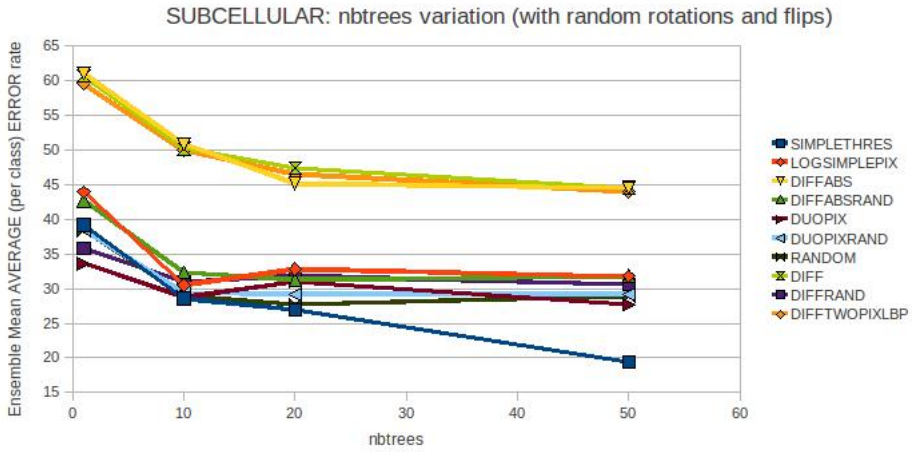


FIGURE A.18: SUBCELLULAR : Erreur moyenne sur les tests simples avec variation du nombre d'arbres et rotations/flips aléatoires en classification directe

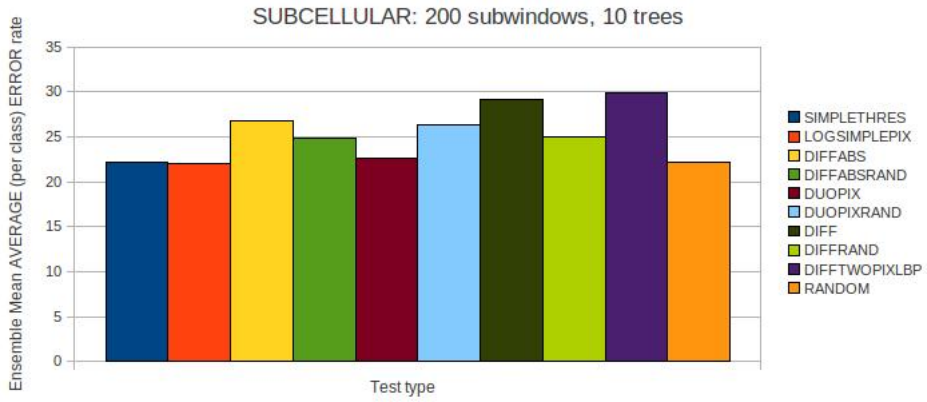


FIGURE A.19: SUBCELLULAR : Erreur moyennesur les différents tests en mode BAGS

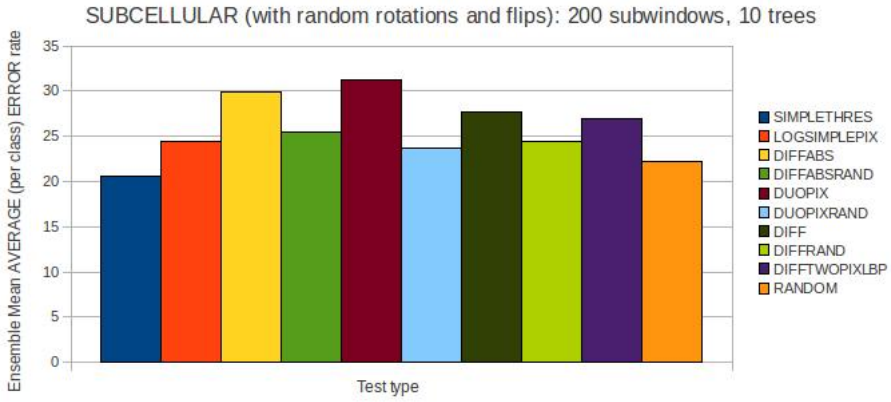


FIGURE A.20: SUBCELLULAR : Erreur moyenne sur les différents tests avec rotations/flips aléatoires en mode BAGS

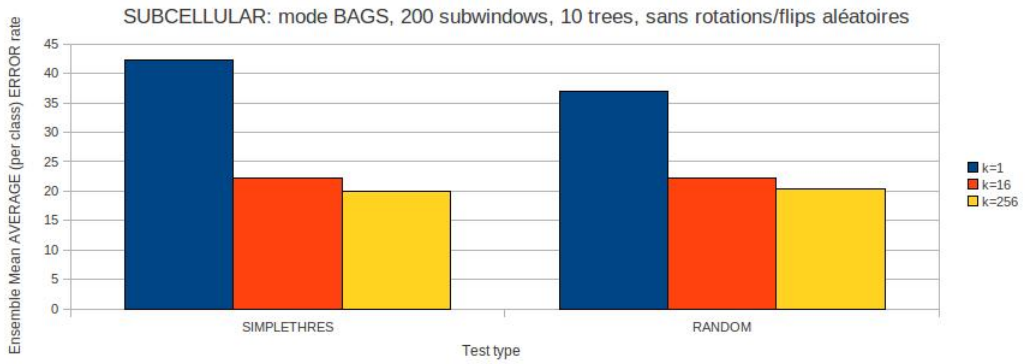


FIGURE A.21: SUBCELLULAR : Variation de k, sans rotations et flips

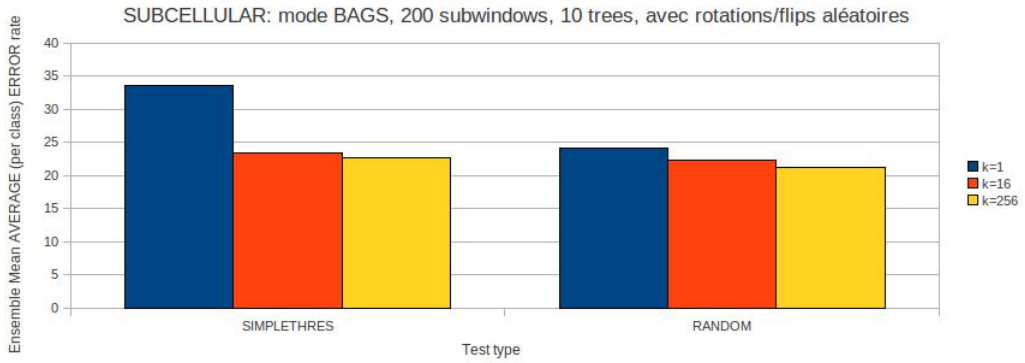


FIGURE A.22: SUBCELLULAR : Variation de k, avec rotations et flips

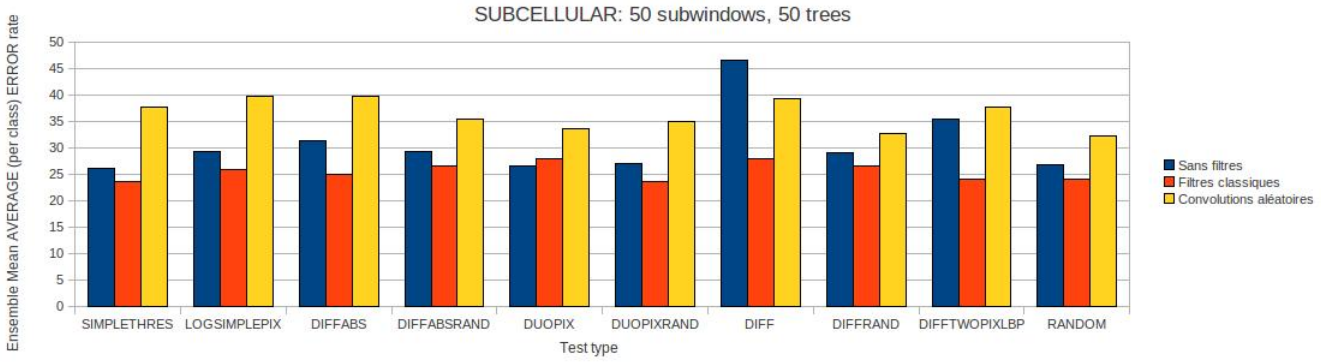


FIGURE A.23: SUBCELLULAR : Filtres sans rotations et flips

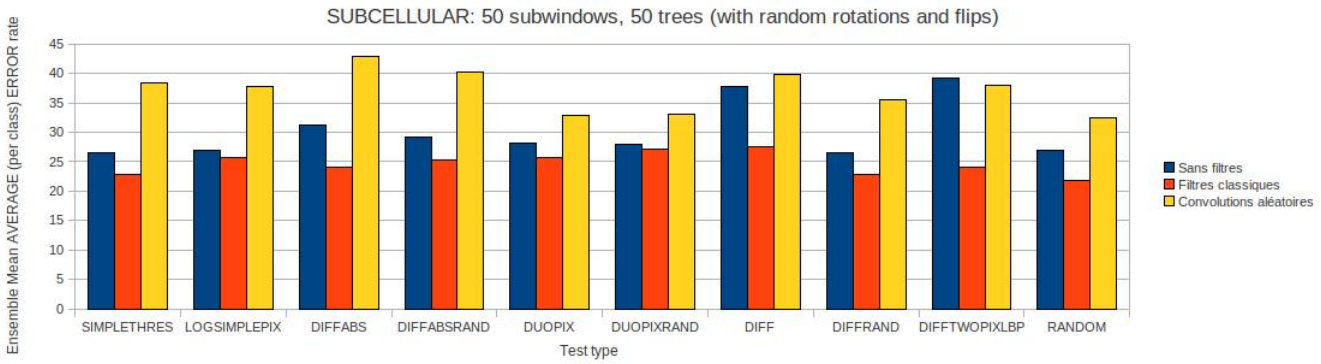


FIGURE A.24: SUBCELLULAR : Filtres avec rotations et flips

A.3 RBC

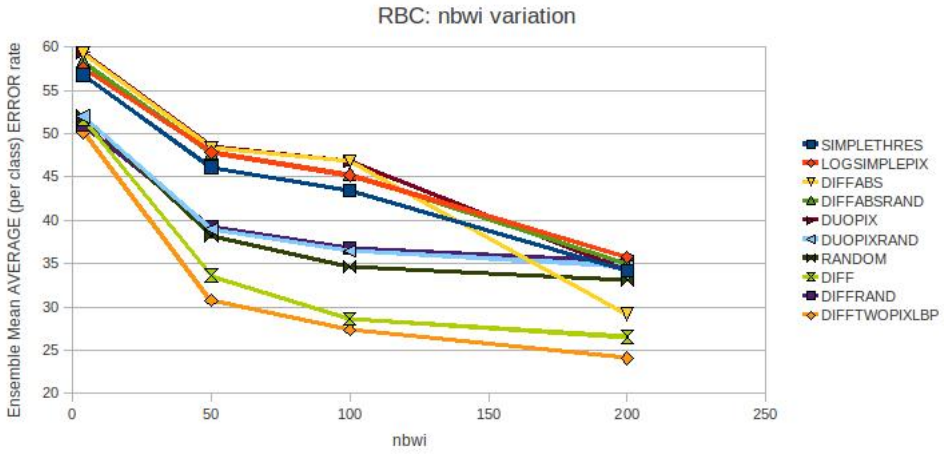


FIGURE A.25: RBC : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres en classification directe

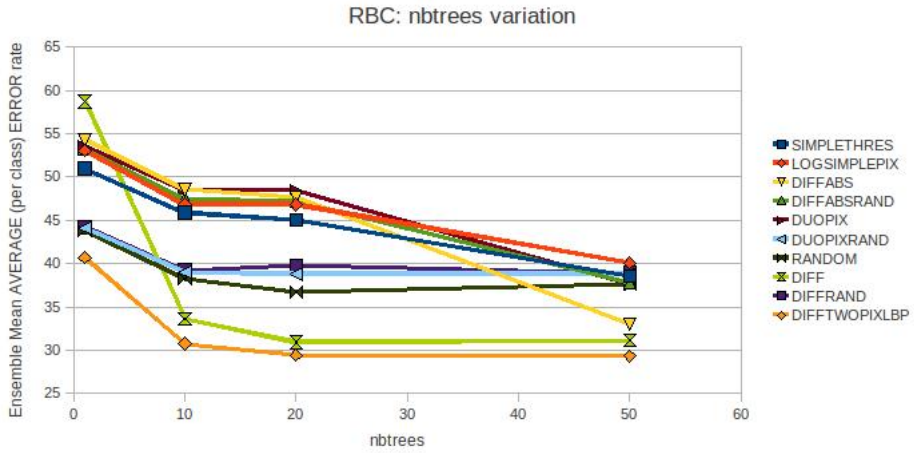


FIGURE A.26: RBC : Erreur moyenne sur les tests simples avec variation du nombre d'arbres en classification directe

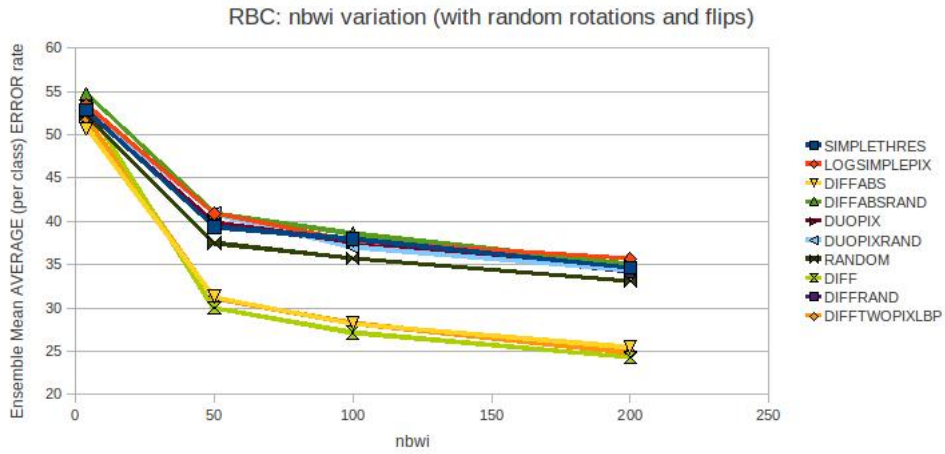


FIGURE A.27: RBC : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

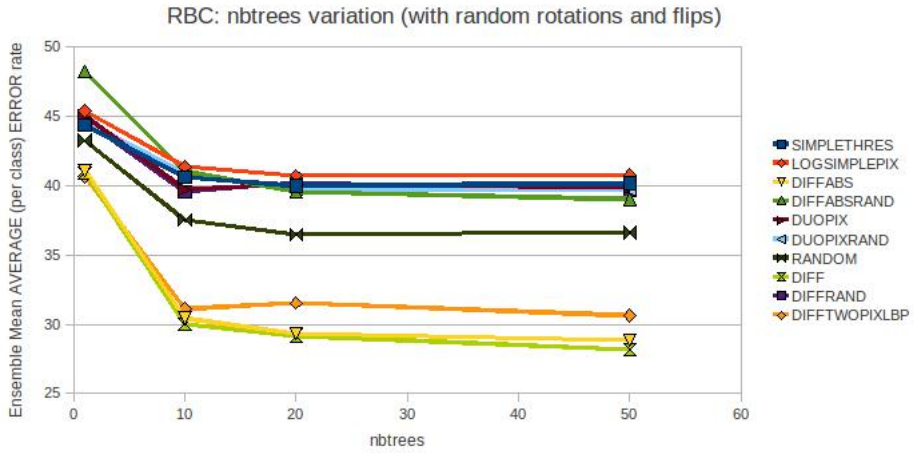


FIGURE A.28: RBC : Erreur moyenne sur les tests simples avec variation du nombre d'arbres et rotations/flips aléatoires en classification directe

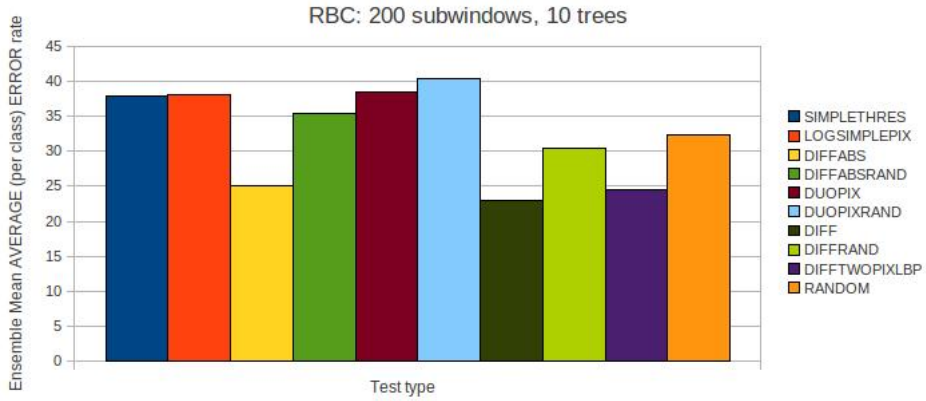


FIGURE A.29: RBC : Erreur moyenne sur les différents tests en mode BAGS

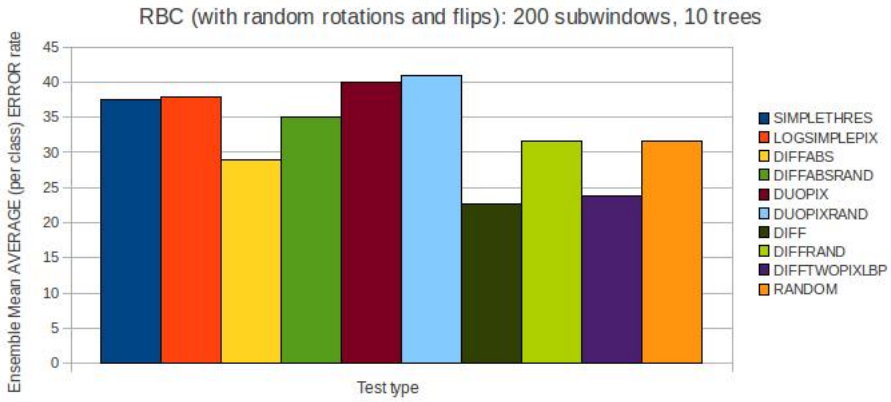


FIGURE A.30: RBC : Erreur moyenne sur les différents tests avec rotations/flips aléatoires en mode BAGS

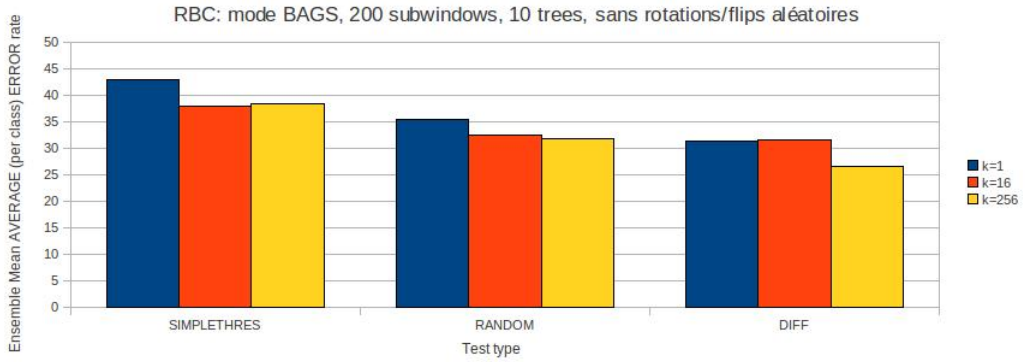


FIGURE A.31: RBC : Variation de k, sans rotations et flips

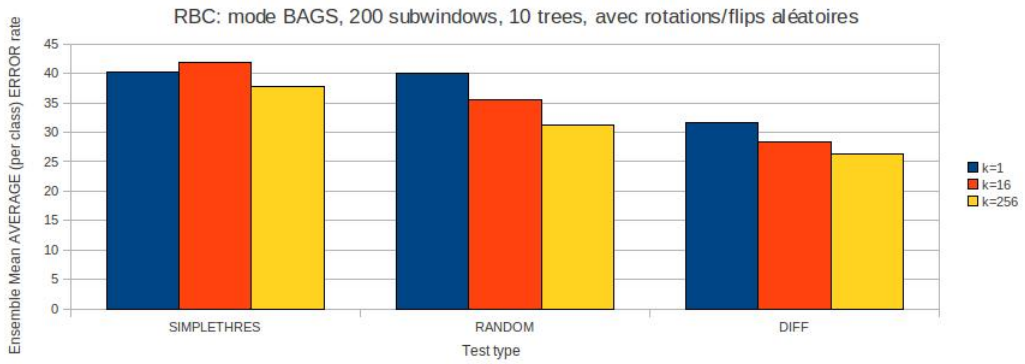


FIGURE A.32: RBC : Variation de k, avec rotations et flips

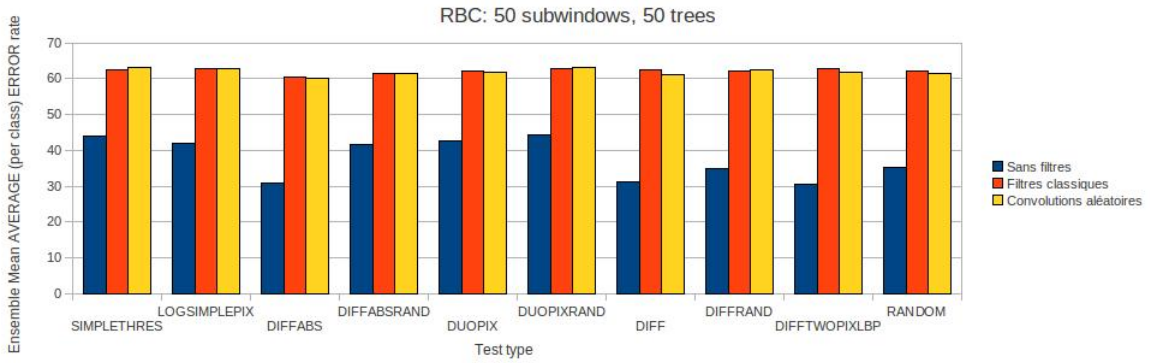


FIGURE A.33: RBC : Filtres sans rotations et flips

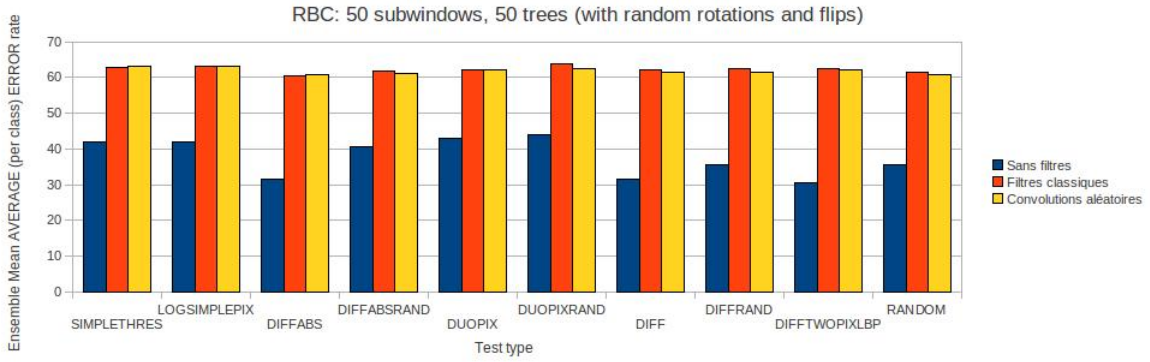


FIGURE A.34: RBC : Filtres avec rotations et flips

A.4 CRYSTAL

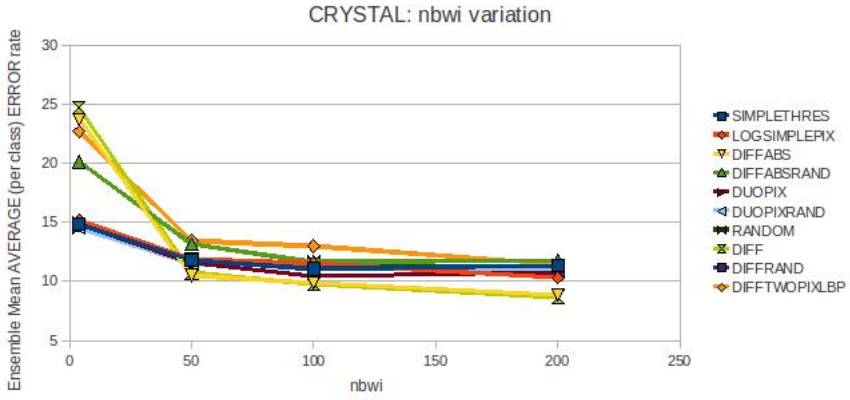


FIGURE A.35: CRYSTAL : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres en classification directe

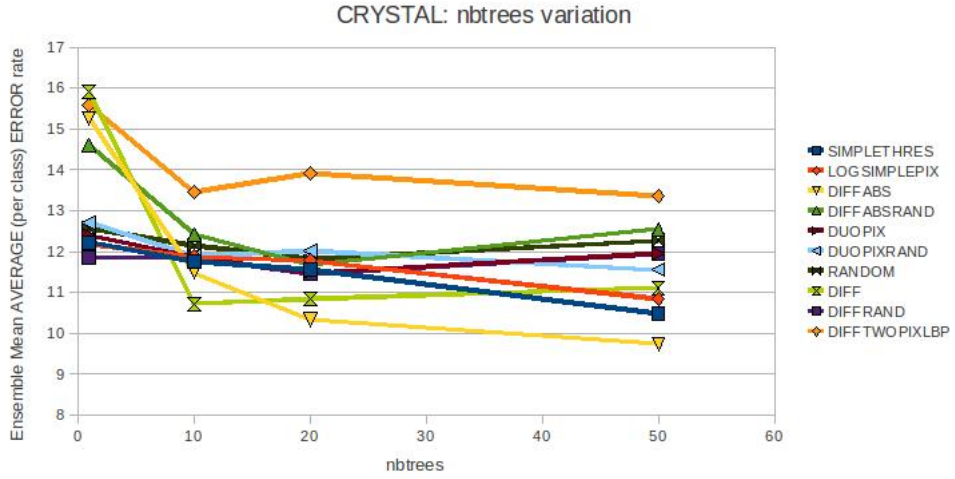


FIGURE A.36: CRYSTAL : Erreur moyenne sur les tests simples avec variation du nombre d'arbres en classification directe

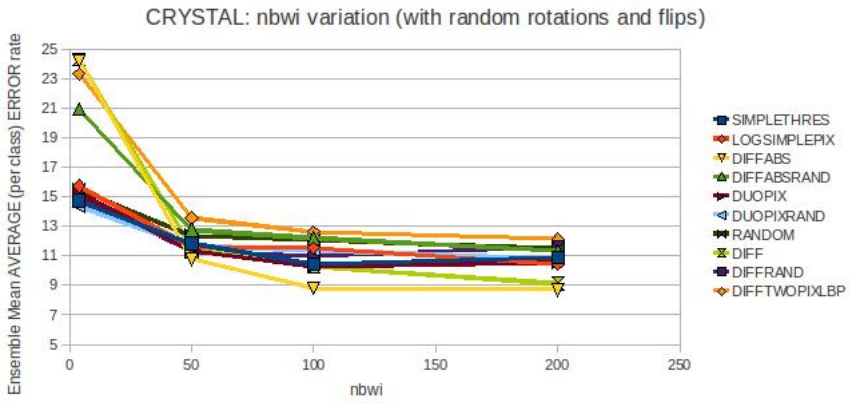


FIGURE A.37: CRYSTAL : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

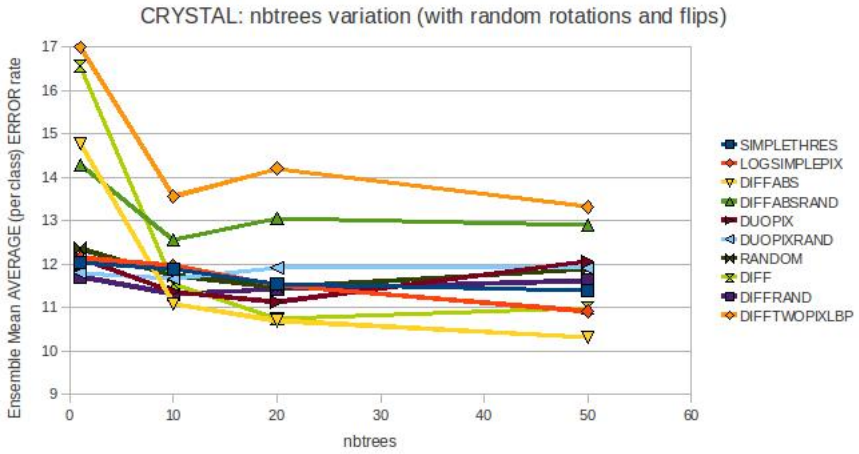


FIGURE A.38: CRYSTAL : Erreur moyenne sur les tests simples avec variation du nombre d'arbres et rotations/flips aléatoires en classification directe

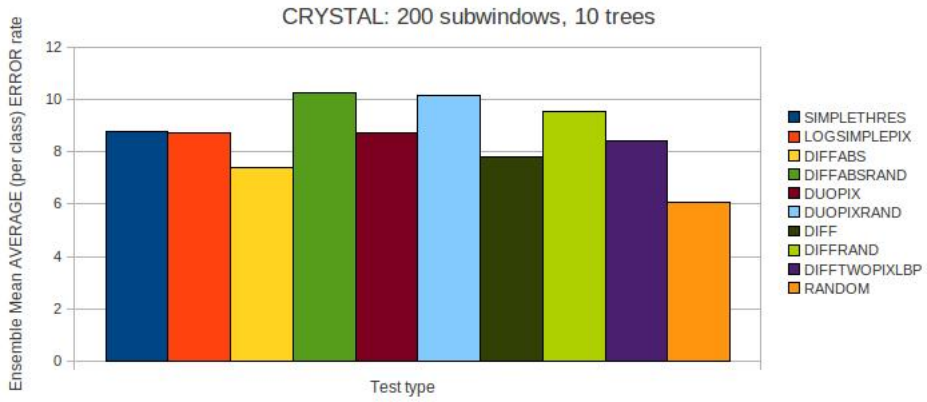


FIGURE A.39: CRYSTAL : Erreur moyenne sur les différents tests en mode BAGS

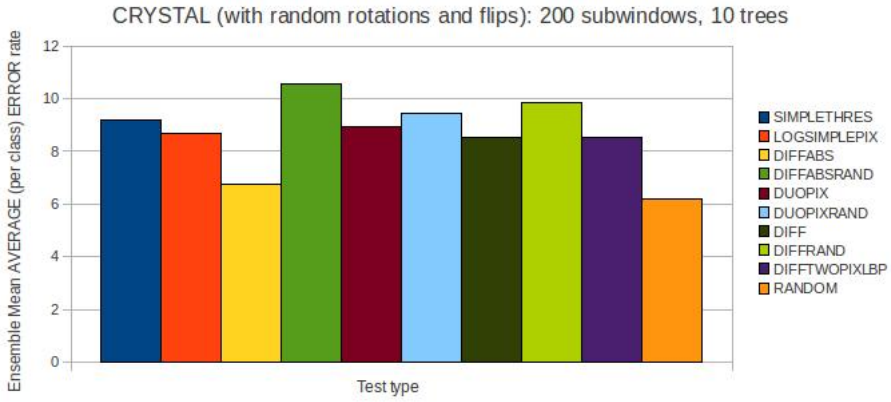


FIGURE A.40: CRYSTAL : Erreur moyenne sur les différents tests avec rotations/flips aléatoires en mode BAGS

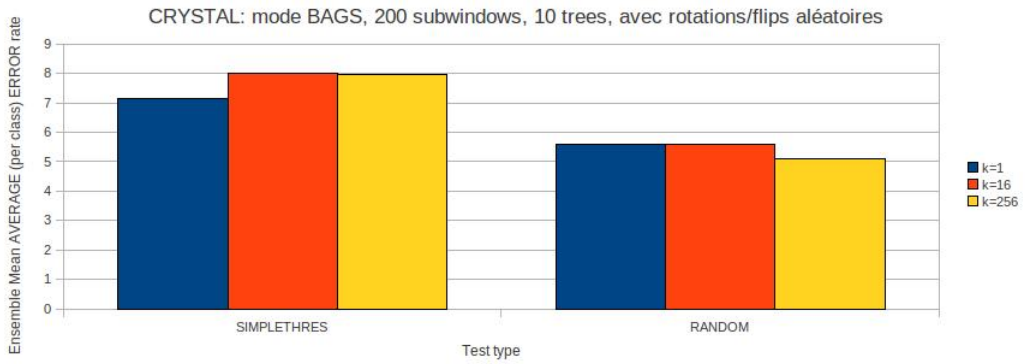


FIGURE A.41: CRYSTAL : Variation de k, sans rotations et flips

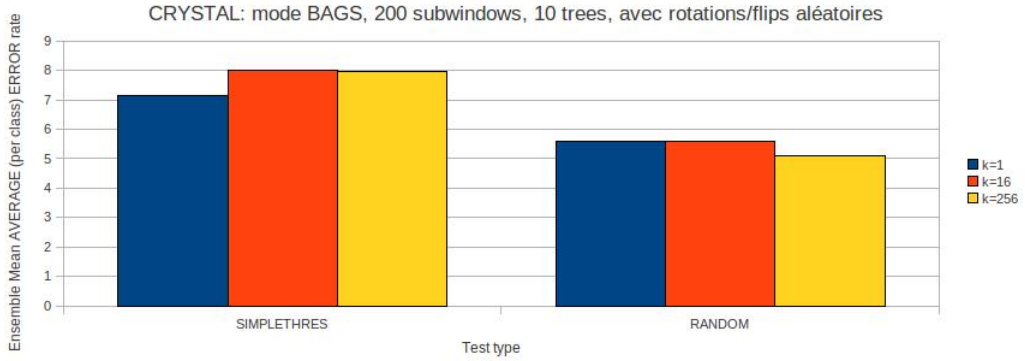


FIGURE A.42: CRYSTAL : Variation de k, avec rotations et flips

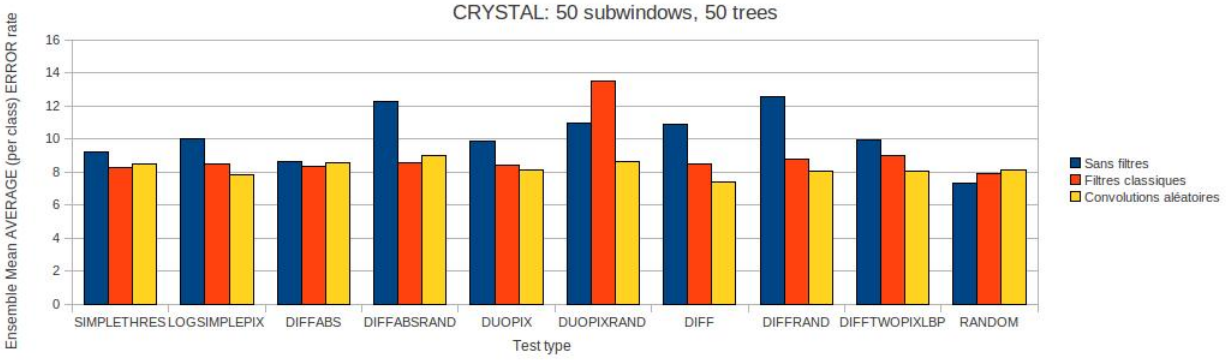


FIGURE A.43: CRYSTAL : Filtres sans rotations et flips

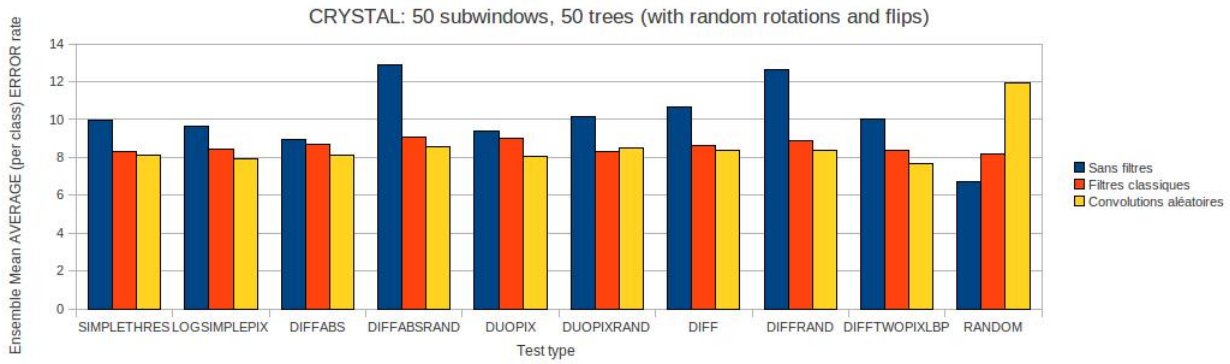


FIGURE A.44: CRYSTAL : Filtres avec rotations et flips

A.5 IRMA2005

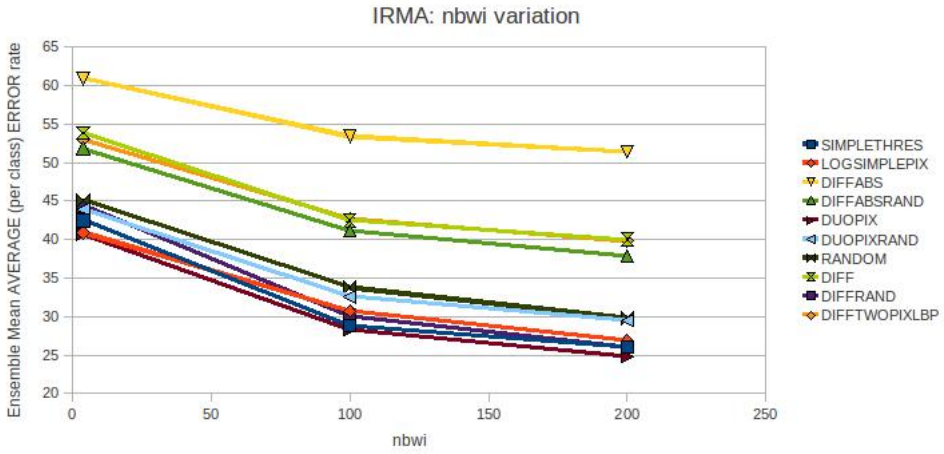


FIGURE A.45: IRMA : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres en classification directe

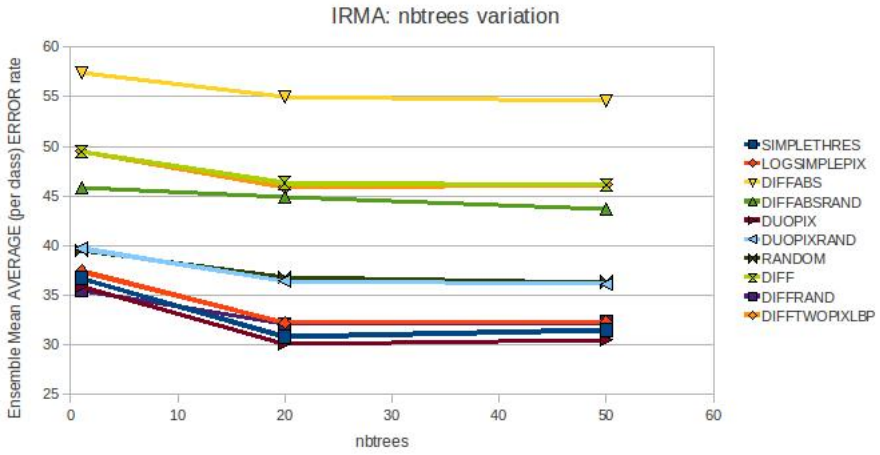


FIGURE A.46: IRMA : Erreur moyenne sur les tests simples avec variation du nombre d'arbres en classification directe

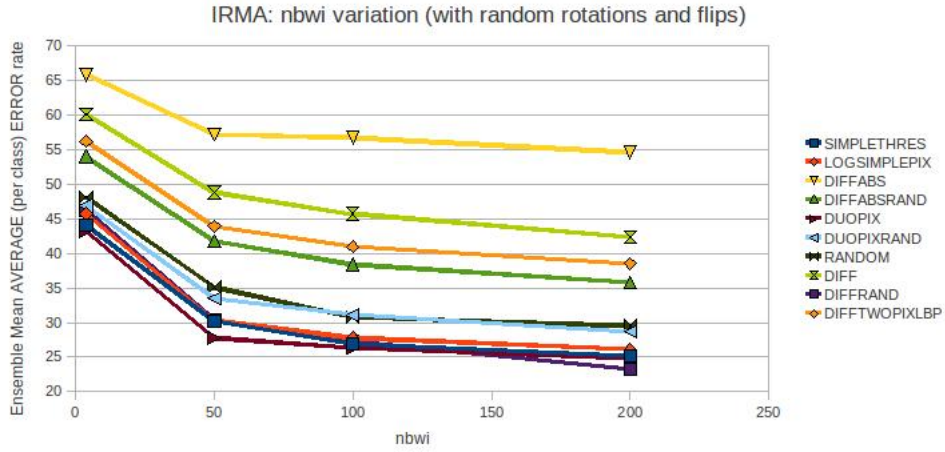


FIGURE A.47: IRMA : Erreur moyenne sur les tests simples avec variation du nombre de sous-fenêtres et rotations/flips aléatoires en classification directe

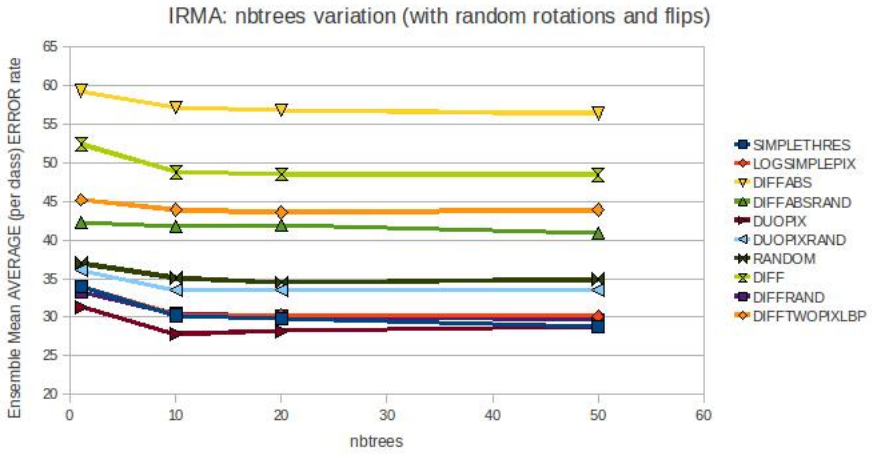


FIGURE A.48: IRMA : Erreur moyenne sur les tests simples avec variation du nombre d'arbres et rotations/flips aléatoires en classification directe

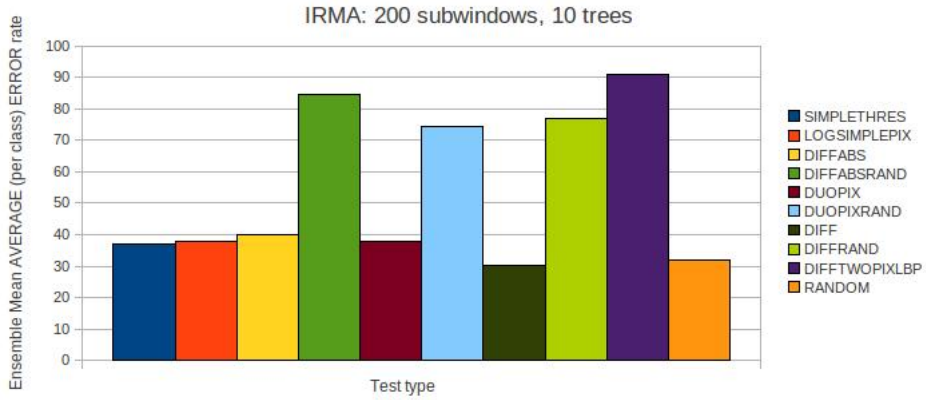


FIGURE A.49: IRMA : Erreur moyenne sur les différents tests en mode BAGS

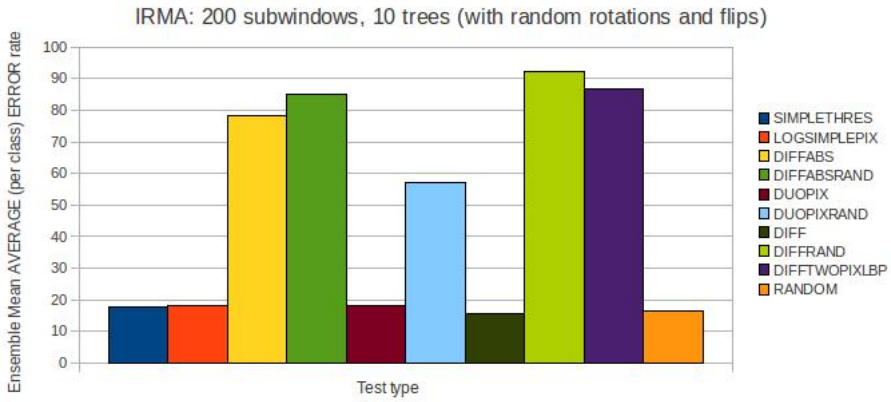


FIGURE A.50: IRMA : Erreur moyenne sur les différents tests avec rotations/flips aléatoires en mode BAGS

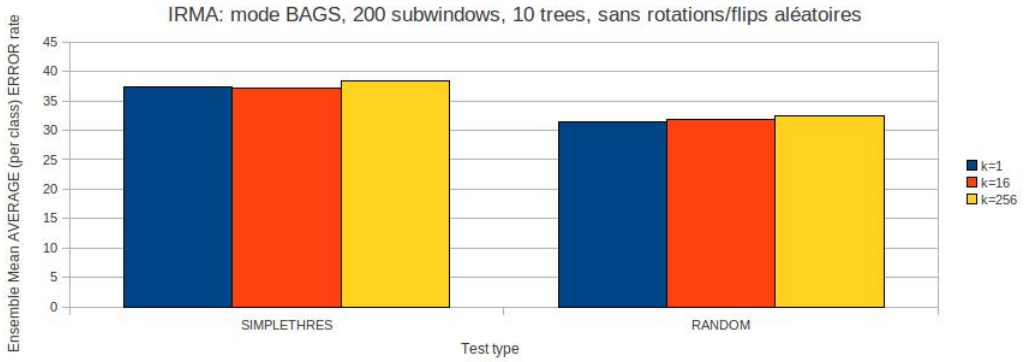


FIGURE A.51: IRMA : Variation de k, sans rotations et flips

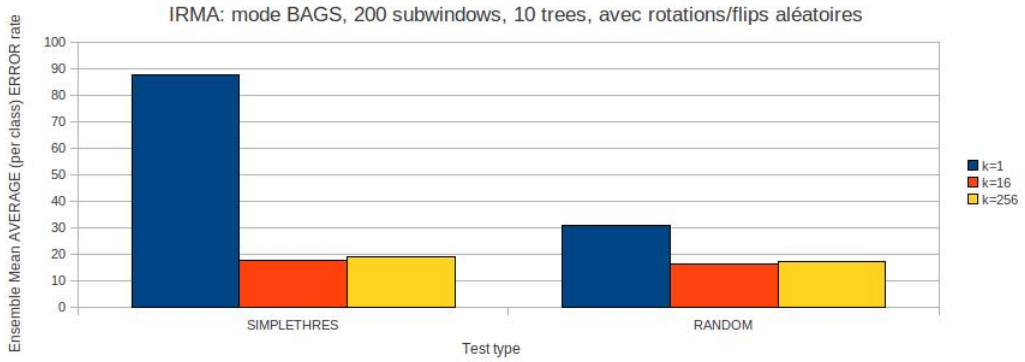


FIGURE A.52: IRMA : Variation de k, avec rotations et flips

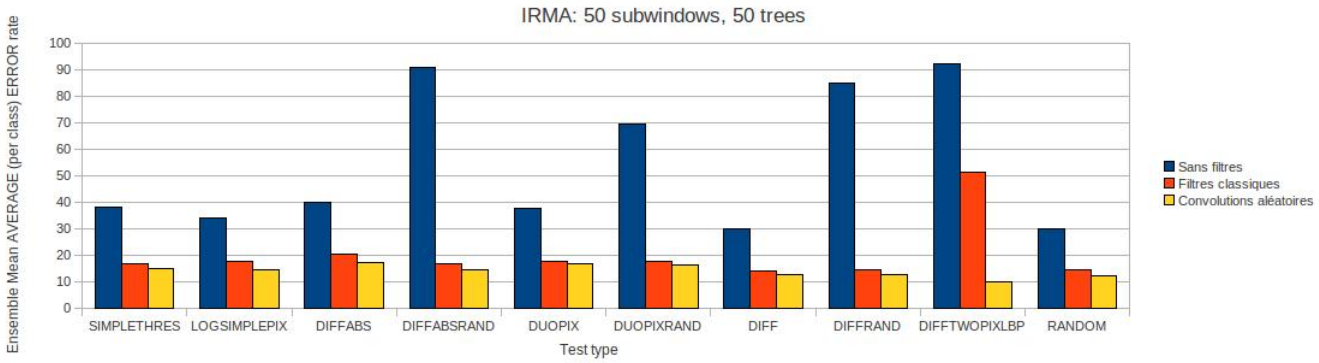


FIGURE A.53: IRMA : Filtres sans rotations et flips

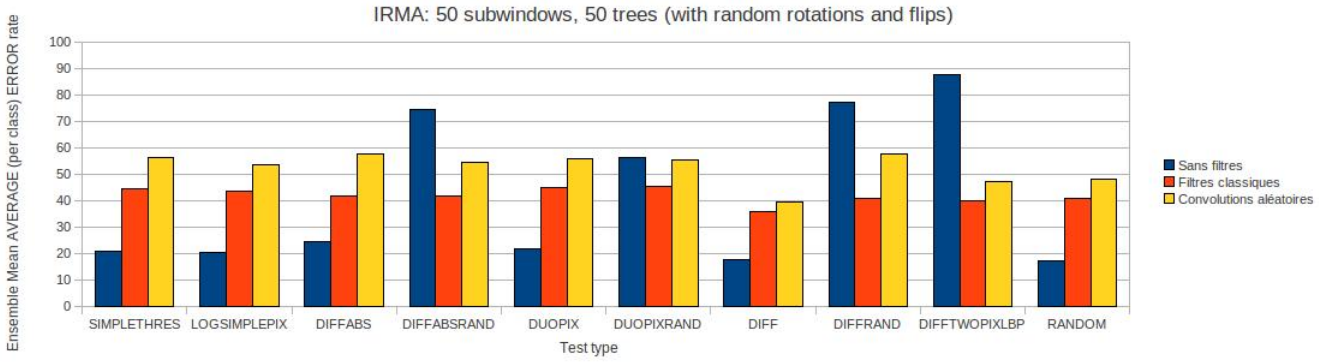


FIGURE A.54: IRMA : Filtres avec rotations et flips

Bibliographie

[ABM07] A. Zisserman A. Bosch and X. Munoz. Representing shape with a spatial pyramid kernel. *CIVR*, 2007.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[Bre01] L. Breiman. Random forests. *Machine Learning*, 45 :5–32, 2001.

[CDF+04] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63 :3–42, 2006. 10.1007/s10994-006-6226-1.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[JZS07] S. Lazebnik J. Zhang, M. Marszalek and C. Schmid. Local features and kernels for classification of texture and object categories : A comprehensive study. *IJCV*, 2007.

[KS07] A. Kumar and C. Sminchisescu. Support kernel machines for object recognition. *ICCV*, 2007.

[LB94] Y. LeCun and Y. Bengio. word-level training of a handwritten word recognizer based on convolutional neural networks. In IAPR, editor, *Proc. of the International Conference on Pattern Recognition*, volume II, pages 88–92, Jerusalem, October 1994. IEEE.

[LF06] V. Lepetit and P. Fua. Keypoint Recognition using Randomized Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9) :1465–1479, 2006.

[Mar05] Raphaël Marée. *Classification automatique d'images par arbres de décision*. PhD thesis, Université de Liège, 2005.

[Mar09] Raphaël Marée. Computer vision and machine learning for the exploitation of biomedical images. Aachen, October 2009. Fraunhofer IME.

[MGPW05] Raphaël Marée, Pierre Geurts, Justus Piater, and Louis Wehenkel. Random subwindows for robust image classification. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1 :34–40, 2005.

[MMvdW07] H. Harzallah M. Marszalek, C. Schmid and J. van de Weijer. Learning object representations for visual object class recognition. *Visual Recognition Challenge*, 2007.

[MNJ08] F. Moosmann, E. Nowak, and F. Jurie. Randomized Clustering Forests for Image Classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(9) :1632–1646, July 2008.

[MSG10] Raphaël Marée, Olivier Stern, and Pierre Geurts. Biomedical imaging modality classification using bags of visual and textual terms with extremely randomized trees : Report of imageclef 2010 experiments. In *CLEF (Notebook Papers/LABs/Workshops)*, 2010.

[Mur] 2D HeLa Images. `http://murphylab.web.cmu.edu/data/`.

[OBI08] Eli Shechtman Oren Boiman and Michal Irani. In defense of nearest-neighbor based image classification. *Computer Vision and Pattern Recognition*, pages 1 – 8, 2008.

[OCV99] Patrick Haffner Olivier Chapelle and Vladimir Vapnik. Svms for histogram-based image classification. *IEEE Transactions on Neural Networks*, 10 :1055 – 1064, 1999.

[OPM01] Timo Ojala, Matti PietikÃ€inen, and Topi MÃ€enpÃ€Ã€. A generalized local binary pattern operator for multiresolution gray scale and rotation invariant texture classification. In *in Proc. of Second Inter. Conf. on Advances in Pattern Recognition, Rio de Janeiro*, pages 397–406, 2001.

[PVG01] E. Poisson and C. Viard-Gaudin. Réseaux de neurones à convolution et reconnaissance de l'écriture manuscrite non contrainte. *Journées VALGO*, pages 726–730, 2001.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1 :81–106, 1986.

[RMW03] Giorgio Visimberga Justus Piater Raphaël Marée, Pierre Geurts and Louis Wehenkel. A comparison of generic machine learning algorithms for image classification. *Proceedings of the 23rd SGAI international conference on innovative techniques and applications of artificial intelligence*, pages 169–182, 2003.

[Weh10] Louis Wehenkel. Cours d'apprentissage inductif appliqué (elen0062-1), leçon 3 : arbres de décision et de régression : principes généraux. Université de Liège, 2010.

[YLF07] T. Liu Y. Lin and C. Fuh. Local ensemble kernel learning for object category recognition. *CVPR*, 2007.