Revue de plusieurs méthodes pour la reconnaissance d'images floues

Rémy Carpentier
Génie de la production automatisée
École de technologie supérieure
Montréal, Canada
remy.carpentier.1@ens.etsmtl.ca

***Résumé* — La mise au point est un critère fondamental pour la photographie, pour la vision par ordinateur et pour toutes les disciplines qui utilisent des images numériques. Le flou est la résultante d'une mise au point mal faite et vient dégrader la photo. Dans ce papier, nous avons comparé 7 méthodes provenant de différents articles pour détecter le niveau de floues dans une image. Deuxièmement, nous avons utilisé 4 classificateurs différents afin de trouver la combinaison caractéristique / classificateur le plus performant possible. Notre objectif final est de proposer un programme basé sur la meilleure combinaison, capable de trier des photographies faites par n'importe quel utilisateur avec n'importe quelle caméra grand public selon leur niveau de floues.**

Mots clés—Reconnaissance de forme, image floue, vision par ordinateur, comparaison

I. INTRODUCTION

A. Faits saillants

La majeure partie des articles traitant de vision par ordinateur, plus particulièrement le flou dans les images, utilisent des méthodes, des algorithmes dits de base sur lesquels les auteurs proposent de rajouter d'autres algorithmes encore plus complexes. Nous avons remarqué qu'ils n'utilisaient pas tous les mêmes algorithmes de bases. Nous avons donc voulu faire un comparatif des méthodes les plus populaires dans la littérature.

Dans ce papier, nous proposons une comparaison de 7 extracteurs de caractéristique et de 4 classificateurs. Nous avons réalisé donc 28 combinaisons différentes d'algorithmes capables de distinguer si une photo est nette ou floue. L'ensemble des résultats vous sera présenté à la fin de cet article. Les différents algorithmes dont nous nous sommes servis n'ont été peu voir pas modifiée pour pouvoir comparer l'essence même de leur méthode. Nous avons utilisé le logiciel Matlab pour ce projet. Deuxième point, grâce à ce comparatif, nous avons pu trouver la meilleure combinaison. Nous l'avons réutilisé dans un programme capable de trier et de classer des photos selon leur niveau de floues.

B. Mise en contexte

La photographie est une technologie vieille de plus d'un siècle. Pourtant son fonctionnement est toujours le même depuis le début : exposer un capteur argentique ou numérique a une scène lumineuse à travers un système optique, la lentille, durant un très court instant. Durant ce court instant, de l'ordre de la centième de seconde en moyenne, si le sujet photographie bouge ou si la caméra bouge un flou va apparaitre. C'est-à-dire que l'image ne sera pas nette, les contours des formes seront mal définis. Le flou peut être voulu dans certains cas. Le photographe peut faire la mise au point sur son sujet et obtenir un arrière-plan flou. De même pour un sujet en mouvement, si le photographe suit son sujet, il va y avoir un flou de bouge sur l'arrière-plan. Toutefois, les images floues peuvent être involontaires et dégrade une photo. C'est le cas si la mise au point n'est pas bien faite, si le sujet bouge ou encore si le photographe tremble pendant la photo entre autres. On peut classer le type d'images selon 5 catégories : les photos nettes (pas de présence de flou), ensuite les photos ayant des parties nettes et d'autres, flous, profondeur de champ, sujet en mouvement, enfin si l'image est totalement floue cela est dû soit à un mouvement de la caméra, soit à une mise au point ratée. L'objectif de notre travail est de réaliser un programme capable de classer des images en deux catégories, nette ou floue.

Fig. 1. *Quatre différents types de flous*

C. *Problématique et objectifs*

Malgré de nombreuses améliorations technologiques dans le domaine, comme l'augmentation de la définition des capteurs et les stabilisateurs optiques et pour le capteur ou encore les modes d'autofocus de plus en plus performants, il est toujours possible de rater une photo pour les raisons citées plus haut. Concernant la détection d'images floues, il existe plusieurs brevets ayant pour but d'intégrer directement dans la caméra ou le téléphone intelligent, un algorithme capable dire si la photo que vous venez de prendre est floue ou pas. Cela évite d'avoir toutes les photos faites floues à cause d'un mauvais réglage par exemple.

Dans cet article, nous allons essayer de répondre à une problématique de moins grande envergure. Nous allons nous focaliser sur le problème suivant : comment réduire le temps qu'un photographe, amateur ou professionnel, passe-t-il à trier et supprimer ses photos floues, que ce soit des photos de vacances, de mariage, ou de n'importe quelle occasion ?

L'objectif majeur est de détecter de façon automatique par ordinateur si une image est floue ou pas. Notre deuxième défi est celui de proposer une solution efficace et rapide. Par rapide, nous entendons inférieurs au temps mis par un humain pour réaliser la même tâche.

Pour un être humain, cette tâche est assez facile. On est capable de comprendre les différents éléments qui constituent la photo, de trouver le sujet important et de dire si celui-ci est flou ou non. Même si, selon les personnes, on peut avoir des avis différents. Nous reviendrons sur ce point dans la section discussions. Pour un ordinateur la tâche est beaucoup plus complexe. Premièrement, il lit une image comme étant est une matrice en trois dimensions correspondant à l'intensité de chaque pixel selon les trois couleurs : rouge, vert et bleu. Deuxièmement, il ne peut pas savoir ce que le photographe a voulu capturer. Il faut donc lui apprendre à détecter du flou et aussi à reconnaitre si ce flou est voulu et s'il est non désire. Nous avons aussi eux d'autres défis comme celui de proposer une solution plus rapide que le traitement manuel et qui propose une solution la plus efficace possible.

Ce document est organisé de la façon suivante : après une revue de la littérature dans le domaine, la prochaine section va présenter les différentes méthodes d'extractions de caractéristiques et les classificateurs étudiés. Ensuite, une section va vous présenter la méthodologie et les résultats obtenus. Enfin, les dernières sections seront consacrées à la discussion des résultats obtenus, aux améliorations pour de futurs travaux et à la conclusion.

II. ÉTAT DE L'ART

P. Hsu et B.-Y. Chen [1] ont proposé une méthode pour pouvoir classifier les images selon 5 catégories : nettes, avec une profondeur de champ, avec un objet en mouvement, avec un mouvement de caméra ou avec le focus mal fait. Pour résoudre ce problème, les auteurs ont utilisé plusieurs niveaux de décision avec plusieurs algorithmes à chaque étape. Ils utilisent le gradient de l'image et un SVM pour trier les photos nettes ou floues. Ensuite pour savoir si une photo est totalement floue ou partiellement, ils utilisent encore une fois le gradient avec simplement un seuil. Enfin pour savoir si le flou est un flou de mouvement, que ça soir l'objet ou la caméra, ils utilisent un estimateur de la fonction de propagation de point (PSF, point spread function en anglais).

R. Liu, Z. Li et J. Jia [2] se sont intéressé a la classification des images partiellement floues. Ils ont proposé une méthode capable de différentier si le flou est dû au mouvement de la camera ou s'il est dû a une mauvaise mise au point. La méthode qu'ils ont utilisée repose sur le découpage de l'image en plusieurs petites parties. Ensuite, ils ont utilisé la pente du spectre de puissance local (Local Power Spectrum Slope), la variance du gradient de l'image (Gradient Histogram Span), le maximum de la saturation et la congruence locale d'autocorrélation (Local Autocorrelation Congruency). Enfin, ils utilisent un classificateur de type Bayesien pour trouver la classe de la photo.

J. Shi, L. Xu et J. Jia [3] ont aussi utilisé les gradients de l'image dans leur travail ainsi que le spectre de l'image dans le domaine fréquentiel. Pour les gradients, ils ont regardé la forme que peut prendre la distribution des valeurs. Dernières méthodes, ils ont utilisé le filtre laplacien pour extraire les caractéristiques. La suite de leur article est sur la segmentation des zones de l'image.

E. Mavridaki et V. Mezaris [4] ont mis au point leur propre méthode basée sur le domaine fréquentiel, le spectre de puissance obtenu avec la transformée de Fourier pour être précis. Ils utilisent un SVM pour classifier les caractéristiques obtenues. Ils ont aussi rendu publique leur base de données. C'est avec cette base de données que nous allons faire nos tests.

R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis et W. T. Freeman [5] ont travaillé sur la déconvolution pour corriger le flou présent dans les images. Leurs caractéristiques principales pour comprendre quel type de flou ils essaient de corriger edst fixer sur la distribution de la magnitude des gradients de l'image. Ils ont aussi utilisé la méthode de l'estimateur de la fonction de propagation de point pour corriger le flou.

B. Su, S. Lu et C. Tan [6] ont travaillé sur la segmentation des zones de floues dans une image. Pour cela, ils ont utilisé la décomposition de la valeur singulière (SVD) et la convolution par l'estimateur de la fonction de propagation. Une fois le flou détecter, ils ont utilisé la fonction canal alpha pour différentier le flou de mouvement et le flou de mise au point.

H. Lee et C. Kim [7] ont aussi proposé une méthode pour la segmentation d'une image selon trois classes, nettes, floues de mouvement et floues de mise au point. Leurs méthodes reposent sur l'amplitude du gradient, sur la cohérence directionnelle et sur un raffinement par région de l'image. Ils utilisent un SVM pour classifier leurs données.

H. Tong, M. Li, H. Zhang et C. Zhang [8] proposent une méthode basée exclusivement sur la transformation d'ondelettes (Wavelet Transform) et sur la présence d'arrêtes dans l'image. Ils proposent une classification selon trois classes de l'image : nette, flou de mouvement et flou de mise au point.

J. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez et J. Fernandes-Valdivia [9] présentent dans leurs articles une façon

très simple pour faire la convolution avec le filtre laplacien. Ils étudient aussi l'impact de la taille de ce filtre sur l'image obtenue et la performance pour catégoriser celle-ci.

S. Pertuz, D. Puig et M. A. Garcia [10] ont fait une revue très intéressante de 36 méthodes différentes existantes et sur leurs performances pour détecter le niveau de flou dans une image. Ils ont utilisé leur propre base de données pour les tests. Les différentes méthodes sont comparées sur leur capacité à retrouver la fonction de dispersion d'un point, soit une seule valeur.

III. METHODOLOGIE

A. *Extraction des caractéristiques*

Il existe un très grand nombre de méthodes et d'algorithmes présents dans la littérature capable de faire ressortir des caractéristiques pour connaitre le niveau de netteté d'une image. Nous en avons sectionné 7 parmi toutes. Nos choix reposent sur plusieurs critères : la fréquence d'apparition d'une méthode dans la littérature, le degré de performance obtenue avec la méthode et aussi son niveau de complexité pour le côté programmation. Nous ne faisons pas une comparaison de toutes les méthodes existantes, mais plutôt une sélection des méthodes les plus communes. On peut remarquer aussi que de nombreux articles font la différence entre 3 voir 5 catégories différentes. Pour notre travail, nous nous sommes contentées de deux classes seulement. La principale raison est le manque de base de données avec 3 ou 5 labels différents.

Dans cette section, nous allons présenter le fonctionnement en détails des 7 extracteurs de caractéristiques que nous avons utilisées. Leurs noms sont le nom donné au script que nous avons écrit dans Matlab.

À savoir que nous avons fait un prétraitement sur toutes les images de notre base de données. Nous avons converti toutes les images en noir et blanc et redimensionnées à 800x600 pixels. Cela permet principalement de pouvoir plus rapidement les images sans perdre trop d'informations sur leur contenu.

1) *Conv*

Le premier extracteur de caractéristique présenté ici est *Conv*. Il repose sur la convolution et le filtre de Laplace. Nous avons pris notre image d'un côté, le filtre de Laplace de l'autre. Ensuite, nous avons fait la convolution de ces deux matrices. Tout d'abord, le filtre laplacien s'écrit de la façon suivante :

$$\nabla^2 = \frac{4}{(\alpha+1)}\begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix} \quad (1)$$

Nous avons donc un paramètre pour cet extracteur, α. Pour le fixer efficacement, nous avons coupé notre base de données en deux tiers, un tiers et ensuite regarder l'entrecroisement des données. Nous avons pu ainsi régler le paramètre sans toucher aux données de tester et éviter le sur apprentissage. Les différentes valeurs obtenues sont présentées dans la section résultats.

Deuxième point dans cette méthode, la convolution. On va venir balayer l'image avec le filtre ci-dessus et obtenir une image différente de l'original. Pour cela, on va venir multiplier le filtre laplacien avec une partie de notre image. Le résultat de cette multiplication est noté dans le pixel au centre de la fenêtre. On recommence cette opération pour chacun des pixels de l'image.

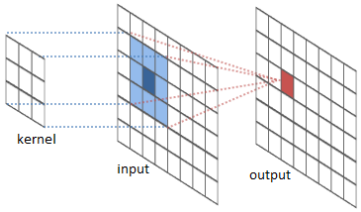

Fig. 2. *Exemple d'une convolution*

Comme sur l'image ci-dessus, cette méthode permet de faire ressortir les contours des formes présentes dans l'image. Si l'image est nette alors ses contours le seront aussi et vont bien ressortir sur la convolution et inversement pour les images floues. Pour réduire la dimensionnalité de cette méthode, nous avons décidé de faire la variance sur chacune des colonnes de l'image. Comme nous l'avons dit, si une image est nette, ses contours le seront aussi, sa convolution sera très contrastée et la variance sera élevée. À l'inverse une image floue aura une variance faible. Nous obtenons donc un vecteur de caractéristique de la taille de l'image après la convolution.

Fig. 3. *Convolution avec un filtre laplacien*

2) *FFT*

La méthode FFT, Fast Fourrier Transformation, utilise le domaine fréquentiel. C'est-à-dire que cette méthode convertit une image deux dimensions en une sorte d'histogramme avec les valeurs le plus fréquentes. On retrouve souvent dans la littérature les transformées FFT sous la forme d'une image noir et blanc comme celle qui est ci-dessous. On peut voir que les sortes de traits sur la FFT représentent les lignes droites sur l'image de gauche et les formes de gamma représentent les courbes de l'image.

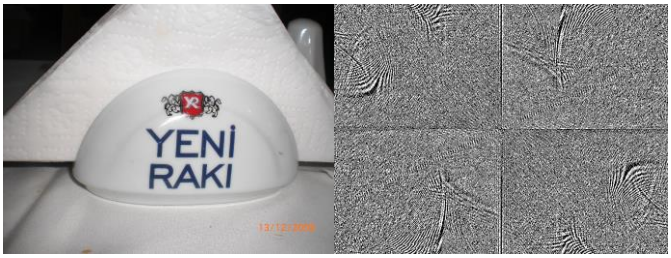

Fig. 4. *FFT (à gauche image originale, à droite le résultat FFT)*

Le résultat obtenu contient toute l'information présente dans l'image originelle et il est possible de faire une transformée inverse pour retrouver l'image. Pour ce qui nous concerne, encore une fois, nous avons dû déduire la dimension des caractéristiques. Pour réduire la FFT, nous avons pris ses valeurs absolues, ensuite nous avons pris le logarithme de ces valeurs et enfin, nous avons fait la somme sur les colonnes des valeurs obtenues. Grâce à cela, nous avons un vecteur de caractéristique de 600 valeurs. Étant donné que les valeurs obtenues sont symétriques, nous avons divisé en deux nos caractéristiques pour n'en garder que 301.

Encore une fois, si une image est nette, il y a de fortes chances que l'on voit apparaitre des motifs comme sur l'exemple ici. Au contraire, une image floue aura pixels ayant peu de valeurs différentes et donc la FFT obtenue sera sans motif singulier.

3) Hist

Le fonctionnement de notre troisième méthode est très simple. Elle repose uniquement sur l'histogramme. L'histogramme en photographie noir et blanc correspond à la répartition du nombre de pixels en fonction de leur intensité de gris. En d'autres termes, un pixel peut avoir une valeur entre 0 et 255. On comptabilise le nombre de pixels égal à 0 puis à 1 jusqu'à 255. On obtient alors un vecteur de 256 valeurs.

Il n'y a pas de lien entre la répartition des données dans les 256 classes et le flou présent dans l'image. Intuitivement, nous savons que cette méthode n'allait pas performer aussi bien que les autres, mais sa simplicité nous a permis de rapidement l'implanter dans notre code et nous avons pu vérifier si nos sentiments étaient vrais ou faux.

4) HOG

HOG, pour Histogram Oriented Gradient, soit en français histogramme de gradient orienté, est une méthode qui repose sur le gradient de l'image. Pour calculer le gradient d'une image, on multiplie chaque pixel par un noyau permettant de faire ressortir les lignes verticales et horizontales. Le filtre que nous utilisé est le filtre de Sobel.

-1	0	1

-1
0
1

Fig. 5. *Filtre de Sobel utilisé pour le gradient*

Les deux images obtenues sont alors les dérivés. Ensuite il faut calculer l'orientation et l'amplitude du gradient de chaque pixel. On utilise les formules ci-dessous. La première équation correspond aux images obtenues avec le filtre. La deuxième nous donne l'amplitude et la troisième, l'orientation. Enfin, on vient calculer l'histogramme de la distribution de ces deux valeurs. Les caractéristiques sont stockées sous la forme d'un vecteur de la taille de nombre de classe pour les deux histogrammes.

$$\Delta S = \begin{bmatrix} \frac{\partial}{\partial x} S \\ \frac{\partial}{\partial y} S \end{bmatrix} = \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

$$|\Delta S| = \sqrt{S_x^2 + S_y^2} \qquad \theta = \tan^{-1}\left(\frac{S_x}{S_y}\right)$$

Cette méthode repose sur les contours et les contrastes dans une image. Pour notre application, cette méthode est donc plutôt prometteuse. Comme la méthode *Conv*, nous avons dû fixer les paramètres de cette méthode. Pour cela, nous avons coupé en deux la base de test. Les deux paramètres sont le nombre de catégories que peuvent avoir les deux histogrammes. Autrement dit, c'est la taille de notre vecteur de caractéristiques qui change.

5) HOG2

Ici nous avons fait une deuxième version de la méthode du HOG. La raison est que dans le logiciel Matlab, il existe une fonction qui permet de renvoyer directement l'histogramme. Nous avons obtenu des résultats différents avec les deux méthodes. Nous avons donc décidé de garder les deux et de les considérer comme deux méthodes différentes.

Avec cette méthode, on a un paramètre. Notre paramètre est la taille en pixel de la fenêtre dans laquelle on fait une moyenne de résultats obtenus. Cela permet d'éviter de prendre en compte le bruit très important obtenu avec cette méthode pour un seul pixel. Nous avons essayé avec plusieurs valeurs pour le paramètre de cette méthode. La performance est évaluée en fonction du taux de recouvrement des valeurs de la base de test. L'ensemble des tests sont présentés dans la partie résultats.

6) Hough

La transformée de Hough permet de détecter les lignes droites dans une image. Encore une fois, si une image est nette, elle va contenir plus de formes saillantes et un meilleur contraste que dans une image floue.

Nous faisons un peu de préprocessing pour cette méthode. Nous appliquons le détecteur de coin de Canny. Cette méthode nous renvoie une image binaire, avec comme valeur à 1 uniquement les pixels correspondant à des bords. Ensuite on applique la transformée de Hough qui nous renvoie les valeurs de rho et de thêta.

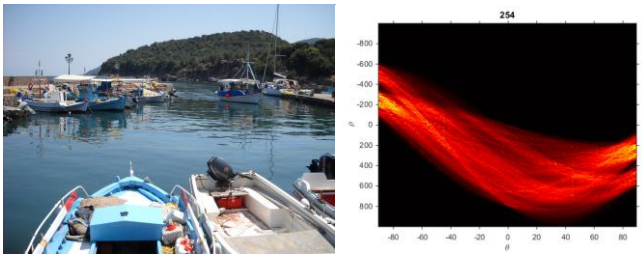

Fig. 6. *Tranformée de Hough*

Ensuite on a décidé de classer les différentes lignes droites obtenues par ordre décroissant et de ne garder que les 100 plus grandes lignes présente l'image. Notre vecteur de caractéristique est donc la norme de chacune des 100 plus grandes lignes droites.

7) WH

La dernière méthode que nous allons vous présenter est basée sur la transformée en ondelette de Haar (WH pour Wavelet Haar). Cette technique est couramment utilisée pour la compression d'image JPEG et aussi en télécommunication. Son principe repose sur le domaine fréquentiel et les transformées de Fourrier.

À partir de la photo originale, on l'a transformé en trois nouvelles images en niveau de gris qui font ressortir respectivement les lignes horizontales, verticales et diagonales. Avec les trois nouvelles images, on vient reproduire la méthode exploitée dans *Hist*, c'est-à-dire que nous avons calculé l'histogramme pour chacune de ces trois images. Ainsi, notre vecteur de caractéristique est les valeurs pour les 256 niveaux de gris pour les trois images soit 768 valeurs. On sait que la méthode de l'histogramme est, intuitivement, non adaptée pour détecter le flou dans les images. Toutefois, le passage dans le domaine fréquentiel peut contrer ce problème.

B. Classificateurs

Nous avons également sélectionné différents classificateurs. De même que les extracteurs de caractéristiques, nous les avons sélectionnés selon leurs performances, leurs popularités et leur degré de complexité. Pour avoir des résultats comparables pour les 28 cas différents étudiés, nous avons mis les résultats obtenus avec ses classificateurs sous la même forme et envoyer dans une unique fonction pour exploiter les résultats. Nous avons utilisé les classificateurs suivants : K-mean, K-nn et SVM. Nous avons réalisé le quatrième classificateur comme étant un mélange des trois premiers. Nous développerons plus dans la partie qui lui est donnée.

1) K-mean

Le classificateur K-mean repose sur le regroupement des données en K groupes. Pour chaque classe, il va donc définir un centre et un rayon dans lequel doivent se situer le plus de points appartenant à ladite classe. On entraine le K-mean sur nos images de tests pour fixer les différents centroïdes.

Pour notre problème de classification, nous avons que deux classes seulement. Nous avons donc fixé la valeur de K à deux pour l'ensemble des tests. Le classificateur est dit non supervisé, c'est-à-dire qu'il n'a pas besoin du label des photos pour le regroupé. Pour calculer le taux d'erreur, on avait soit 15 %, soit 85 % de taux d'erreur avec les caractéristiques du FFT. On a remarqué que le deuxième cas est quand le K-mean inverse les classes et donc on retrouve le 15 % en faisant 1-15 %.

Le K-mean ne donne pas toujours les mêmes résultats dépendamment des points de départ qu'il considère, nous avons fait 5 entrainements différents et conservé celui qui affichait les meilleurs résultats. Aussi, pour sélectionner les ponts de départ, nous avons opté pour la technique qui consiste à faire un premier K-means sur 10 % de la base de données pour prédire l'emplacement des groupes. Donc comme nous l'avons dit, selon les 10 % considéré, nous avons obtenu différents résultats et conservé le meilleur.

Une fois entrainé, on récupère les coordonnées des deux centroïdes pour notre cas. On a ensuite calculé la distance euclidienne des nouvelles valeurs jusqu'aux deux centroïdes. Le centroïde le plus proche donne son label à la nouvelle valeur. On répète l'opération sur l'ensemble des données testé.

2) K-nn

À la différence du K-means, le K-nn est un classificateur supervisé. C'est-à-dire qu'il faut que les données soient labélisées, ce qui est notre cas. Le principe de fonctionnement du K-nn est le suivant, une nouvelle donnée va avoir le même label que ses K plus proches voisins. C'est un classificateur non paramétrique. On a donc un paramètre, la valeur de K. Pour déterminer la meilleure valeur de K, nous avons fait comme pour les caractéristiques. C'est-à-dire, nous avons découper la base de données en deux tiers, un tiers, fait varier K et observer le taux de recouvrement obtenu.

L'avantage de ce classificateur est qu'une fois que le K est fixé, on eut directement passé à la phase de test sans faire d'entrainement. En effet, pour chaque nouvelle image, on va trouver les K plus proches voisins de ses caractéristiques. On utilise la distance euclidienne. Ensuite, on regarde le label le plus fréquent parmi les K voisins et on le donne à la nouvelle image.

3) SVM

Le SVM, pour « support vector machine » en anglais, est un classificateur pour des données labélisées, soit un apprentissage supervisé. Ce classificateur va essayer de trouver les frontières qui permettent de délimiter nos deux classes et de maximiser la marge entre ces classes. Le SVM est un classificateur binaire, ce qui est tout à fait en lien avec notre travail. C'est un classificateur de type séparateur.

Les frontières de décisions dépendent du noyau. Pour ce travail, étant donné que nos données sont assez complexes et non linéairement séparables, nous avons choisi un noyau gaussien. Il faut aussi savoir que ce type de noyau à deux paramètres : C et gamma. Le premier, C, concerne le nombre de vecteurs support : plus il est grand et plus le nombre de vecteurs de support est faible. Le deuxième, gamma, correspond à la variance des données dans les classes : plus gamma est grand, plus la variance est faible. Pour l'ensemble des tests, nous avons fait varier C entre 1 et 1 000 000, par multiplication par 10. Pour chacun des combinaisons que nous avons évaluées avec ce classificateur, nous avons une valeur de C différente. L'ensemble des résultats est développé dans la section correspondante. Pour gamma, nous avons utilisé une option dans la fonction du SVM dans Matlab qui laisse le logiciel trouver la meilleure valeur.

4) All

Le dernier classificateur a été fait par nous-mêmes. Nous avons fait un script qui reprend les trois classificateurs précédents. Pour une image donnée, on a donc trois prévisions. La prévision la plus fréquente est alors sélectionnée et

appliquée à la nouvelle image testée. Cette méthode est plutôt simple, mais permet essentiellement de corriger et d'homogénéiser les résultats obtenus par les trois classificateurs individuellement. L'inconvénient de cette méthode est le temps de calcul. En effet, le temps mis par cette méthode est la somme des temps de chaque classificateur.

C. *Base de données*

Tous les tests ont été faits sur la même base de données nommée « CERTH image blur dataset ». Cette base de données est celle utilisée dans l'article [4]. Elle est constituée de 2450 images numériques prises par différentes caméras et différents photographes. Parmi les 2450 images, 1850 sont de vraies photos en opposition aux 600 restantes qui ont été floutées par ordinateur pour créer plus de données. La réparation originelle est la suivante : 1000 images dans le set d'entrainement (630 nettes, 220 flous naturels et 150 flous artificiels) et 1480 dans le set de test (589 nette, 411 flous naturels et 480 flous artificiels). Nous avons décidé de rééquilibre cette distribution. Notre base d'entrainement est composée comme dans le tableau ci-dessous. Nous avons fait ce changement suite à de meilleures performances lors de nos différents tests. Aussi, rien ne justifiait d'avoir plus d'échantillons pour le test que pour l'entrainement.

Nombre de photos	Lot d'entrainement	Lot de test	Pourcentage
Nette	1000	249	1249 (50 %)
Flou naturel	505	126	631 (25 %)
Flou artificiel	480	120	600 (25 %)
Pourcentage	1985 (80 %)	495 (20 %)	2480

Fig. 7. *Répartition des données*

D. *Programmation*

Étant donné le nombre important de combinaisons que nous avons comparé, nous avons organisé notre travail autour du script *Main*. Ce script va appeler consécutivement 4 autres scripts, *ReadData*, *Choose_Features*, *Choose_Classifier et résultats*. ReadData sert à charger les donnes de toutes les images dans Matlab. Les deuxièmes et troisièmes scripts vont sélectionner l'extracteur de caractéristique et le classificateur que l'utilisateur aura demandé. Enfin, le script *Resultat* se charge d'évaluer les résultats obtenus avec la méthode sélectionnée.

Cette méthodologie nous a permis de rapidement réaliser tous nos tests. Il nous a suffi de modifier uniquement dans le script *Main* les caractéristiques et le classificateur voulu. Autre point intéressant, avec le script *résultat*, nous avons dût normaliser toutes nos données sortantes des différentes méthodes. Ainsi, tous les résultats sont dans les mêmes unités et homogènes.

À noter que pour éviter les problèmes de dimensionnalité, nous avons effectué une analyse en composantes principales. Cela nous a permis de réduire le plus possible la dimension de nos caractéristiques. Certaines caractéristiques ont même dû être modifiées pour avoir un nombre de dimensions raisonnable pour pouvoir faire l'ACP. Notamment la méthode HOG2 qui ressortait avec une dimensionnalité de plus de 50 000 valeurs. Il nous était impossible de faire l'ACP alors.

L'ensemble du travail a été réalisé avec le logiciel Matlab R2015a sur un ordinateur personnel avec un processeur Intel Core i5 (2,6 GHz, 8 GB RAM, Windows 7 64 — bits)

IV. VALIDATION

La section suivante va être consacrée à la présentation des résultats obtenus au cours de nos différents tests. Nous allons aussi expliquer comment les résultats ont été obtenus.

Le premier résultat est le temps nécessaire pour extraire les caractéristiques de toutes les images de la base de données, entrainement et test, soit 2480 photos. Pour simplifier le graphique ci-dessous, nous avons pris la moyenne des temps pour chaque caractéristique. En effet, on a constaté des petits écarts selon le classificateur utilisé.

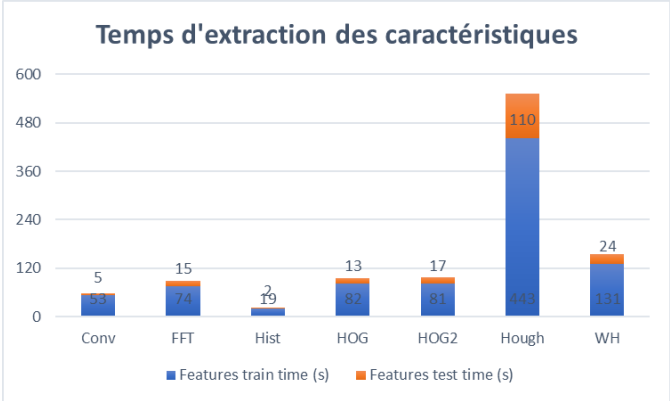

Fig. 8. *Temps d'extraction des caractéristiques*

Le deuxième résultat est le taux d'entrelacement des caractéristiques obtenu. Plus ce taux est faible et moins les caractéristiques se mélangent avec d'autres caractéristiques d'autres labels. C'est-à-dire que les classes sont distinctes. Dans le cas idéal où l'entrelacement serait de 0 %, nous aurions des données linéairement séparables.

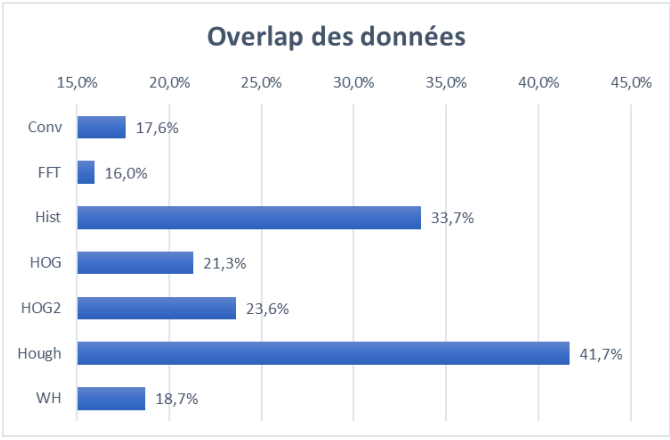

Fig. 9. *Overlap des données*

Aussi si le taux est bas, alors la méthode de réduction de la dimensionnalité ACP sera performante et va diminuer grandement le nombre de caractéristiques utiles. Le nombre de caractéristiques avant et après l'ACP est présenté dans la figure 10. On peut voir que la diminution de la dimensionnalité grâce à la méthode ACP est très importante. On divise par 7 au minimum et 301 au maximum le nombre de dimensions.

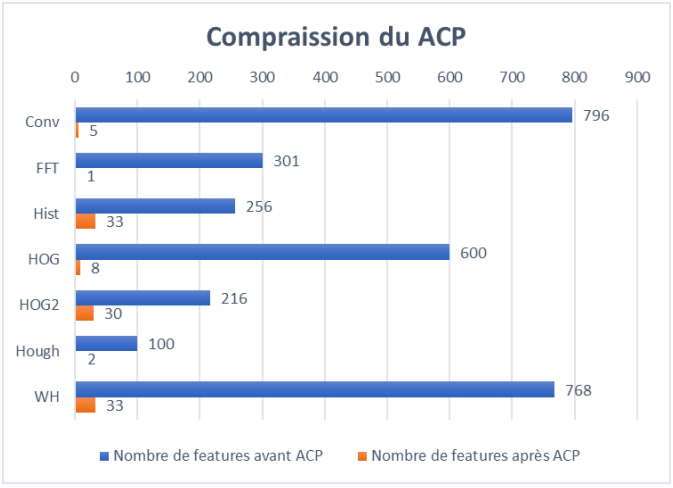

Fig. 10. *Comparaison du ACP*

Le graphique de la figure 11 présente les temps nécessaires pour réaliser l'ACP de chacune des combinaisons. Comme nous l'avons évoqué plus haut, l'ACP n'est affecté que par les différences entre les méthodes d'extraction de caractéristiques et non les classificateurs. On voit bien que les résultats suivent ce raisonnement, mais que les écarts sont assez importants.

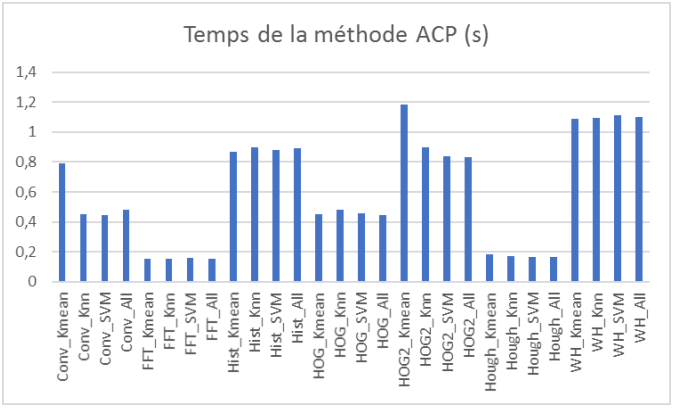

Fig. 11. *Temps de la méthode ACP*

Le tableau ci-contre regroupe l'ensemble des temps nécessaires pour faire l'entrainement et le test pour chacune des 28 combinaisons. Les temps sont affichés en seconde. Le chronomètre est lancé après que les caractéristiques soient extraites et que l'ACP ait été réalisé et arrêté une fois que la dernière photo est traitée par le classificateur. On peut remarquer que les temps sont semblables pour un classificateur, quelles que soient les caractéristiques. On peut aussi observer que le classificateur All est le plus lent, ce qui est logique puisqu'il est la combinaison des trois premiers.

Nom	Train time (s)	Test time (s)	Nom	Train time (s)	Test time (s)	Nom	Train time (s)	Test time (s)
Conv_Kmean	0,86	0,00	HOG_Kmean	0,30	0,00	Hough_Kmean	0,43	0,00
Conv_Knn	1,56	0,02	HOG_Knn	1,70	0,02	Hough_Knn	1,52	0,02
Conv_SVM	76,04	0,84	HOG_SVM	8,88	0,89	Hough_SVM	9,90	1,05
Conv_All	80,42	0,89	HOG_All	10,41	0,90	Hough_All	25,65	1,03
FFT_Kmean	0,27	0,00	HOG2_Kmean	0,40	0,00	WH_Kmean	0,31	0,00
FFT_Knn	1,32	0,02	HOG2_Knn	2,78	0,14	WH_Knn	2,88	0,05
FFT_SVM	140,27	2,27	HOG2_SVM	3,37	1,18	WH_SVM	3,70	0,61
FFT_All	116,64	1,43	HOG2_All	6,35	0,95	WH_All	6,91	0,64
Hist_Kmean	0,34	0,00						
Hist_Knn	3,20	0,05						
Hist_SVM	3,83	0,92						
Hist_All	6,81	1,05						

Fig. 12. *Temps d'entrainement et de test des classificateurs*

Ci-dessous, la quantité de mémoire RAM que chaque combinaison utilise. Les valeurs sont en MB et ont été obtenues avec la fonction « memory » dans le logiciel Matlab. Nous avons mis cette fonction, à la fin de nos scripts pour savoir la quantité globale nécessaire pour chacune des combinaisons.

La figure 14 affiche les valeurs des différents paramètres utilisés. Comme nous l'avons dit plus haut, le classificateur K-nn a un paramètre, le nombre de voisins à prendre en compte soit K. Le noyau gaussien du SVM a deux paramètres, C et gamma. La valeur du premier est affichée dans le tableau alors que gamma est optimiser en interne dans Matlab. Enfin, le classificateur All reprend le K-nn et le SVM. Il a donc leurs deux paramètres, K et C.

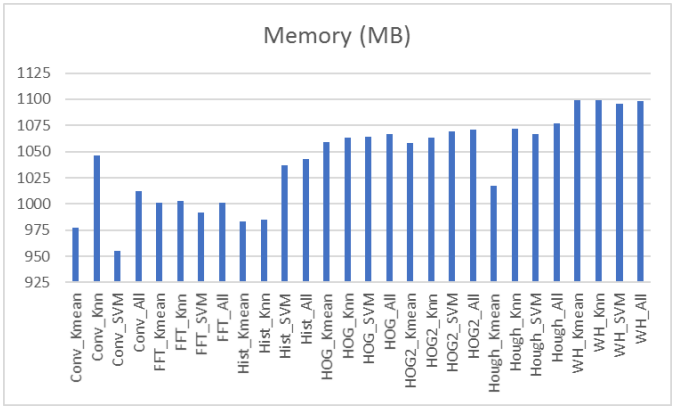

Fig. 13. *Quantité de RAM utilisées par combinaison*

	Knn	Svm	All	
			Knn	SVM
Conv	13	1	48	1
FFT	41	1	25	1
Hist	2	1	1	1
HOG	4	1	10	1
HOG2	8	1	5	1
Hough	31	1	27	1
WH	9	1	6	1

Fig. 14. *Valeurs des paramètres pour les différentes combinaisons*

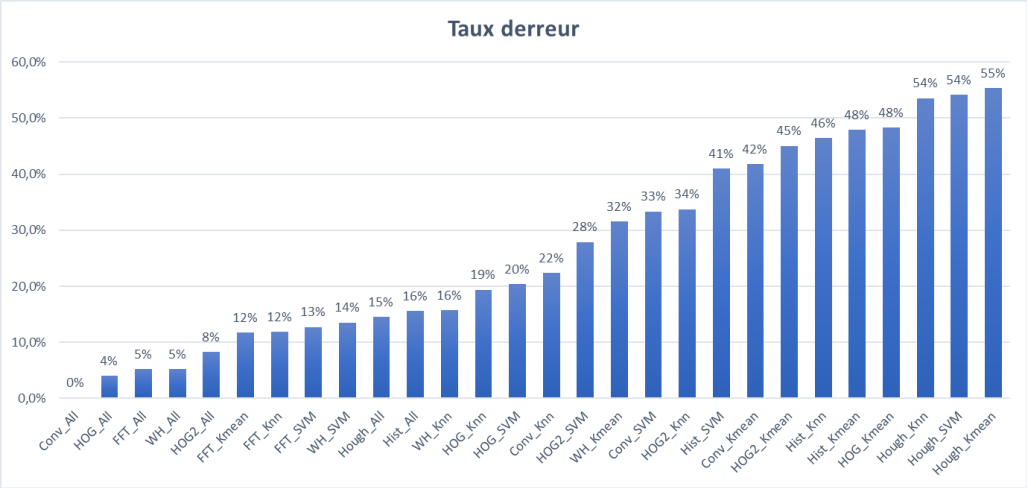

Fig. 15. *Taux d'erreur pour l'ensemble des méthodes*

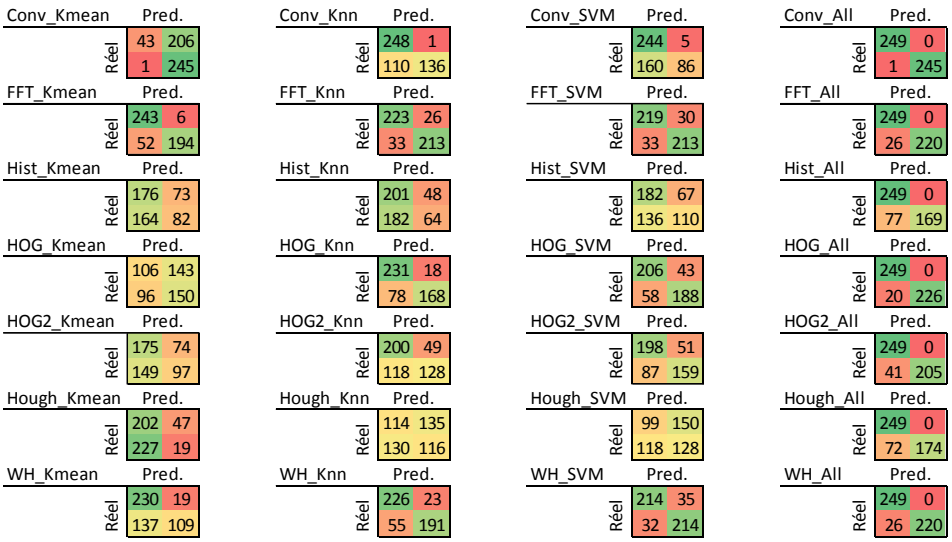

Fig. 16. *Ensemble des matrices de confusions*

La première figure ci-dessus est le taux d'erreur pour chacune des 28 combinaisons que nous avons faites. Pour calculer le taux d'erreur, nous avons utilisé le calcul suivant :

$$Taux\ d'erreur = \left(\frac{nb\ d'images\ mal\ labélisées}{nb\ d'images\ totales}\right) \times 100$$

Nous avons classé les résultats de la plus faible erreur à la plus grande. C'est-à-dire qu'à gauche, on a la méthode la plus performante et à droite celle qui l'est le moins. On peut remarquer aussi que les méthodes avec un taux d'erreur proche de 50 % sont incapables de distinguer une image floue d'une image nette.

La figure 16 est l'ensemble des matrices de confusion obtenues. La matrice se lit comme suit : la première colonne correspond à une prédiction d'image nette et la deuxième colonne, à une prédiction d'image floue. En ligne, nous avons les vrais labels. La première ligne correspond à une image labélisée comme nette et la deuxième ligne à une image labélisée comme floue.

V. DISCUTIONS

La première remarque que nous pouvons faire est que notre classificateur All est le plus performant en termes de taux d'erreur. Il occupe les 5 premières places du classement. Comme nous l'avons dit plus haut, le classificateur All est aussi le plus lent vu qu'il doit réaliser les trois classificateurs avant de donner un résultat. Toutefois, on peut observer que le temps mis lors de la phase de test est de maximum 2,27 seconde, avec All, pour tester 495 photos soit une vitesse de traitement de 5 millième de seconde par photo. La notion de temps est donc négligeable étant donné que le programme que

nous voulions réaliser n'est pas destiné à traiter plus d'un millier de photos à la fois soit un traitement de quelques secondes.

Autre remarque sur la combinaison Conv_All, elle obtient un taux d'erreur de 0,2 %, soit une seule image mal classée sur 495. Cela nous a paru trop bien pour être vrai. De plus, les résultats avec les caractéristiques Conv ne sont pas très performants avec les autres classificateurs (entre 22 % et 42 % d'erreur). Nous avons réalisé 5 essais et nous trouvons toujours ce résultat. Autre point, les résultats confirment bien nos hypothèses de départ, les caractéristiques ayant un rapport avec le domaine fréquentiel sont plus performantes que les méthodes plus simplistes comme l'histogramme, même si ce dernier est le plus rapide grâce à sa simplicité.

Nous avons pu remarquer aussi que les labels de la base de données n'ont pas toujours exactes. Il y a certaines images qui sont floues et labélisées comme nette et inversement. Cela peut être une autre cause des résultats obtenus. Il serait intéressant de voir comment se comportent nos 28 combinaisons si la base de données est parfaitement labélisée, c'est-à-dire sans aucune image mal classée.

Les erreurs de labélisation soulèvent un autre point important. Tout le monde sait ce qu'est le flou. Tout le monde est capable de reconnaitre si une photo est nette ou pas. Toutefois, on a pu remarquer que le seuil à partir duquel une personne classe une image floue ou non est différent pour chacun. On peut parler de seuil de tolérance. Le fait que l'on ait que deux classes n'aide pas. Surtout pour les images ayant une partie nette et une autre floue.

La suite de ce projet serait d'isoler la meilleure combinaison et le d'intégrer dans un petit programme capable d'aller lire les nouvelles images, d'extraire leurs caractéristiques, de faire leur classification et de les renvoyer dans deux dossiers différents, l'un pour les images floues, l'autre pour les nettes. De cette façon, l'utilisateur n'aura plus qu'à vérifier rapidement s'il n'y a pas eu de mauvaises classifications. Si l'on poursuit ce projet, on pourrait regarder les apprentissages semi-supervisés pour éviter d'avoir des erreurs de labels et par exemple demander à l'utilisateur de classer lui-même les photos dont le classificateur n'est pas certain de la classe.

VI. Conclusion

Nous avons vu dans ce travail plusieurs méthodes pour différentier une image nette d'une image floue par reconnaissance de forme. Nous avons étudié 7 extracteurs de caractéristique et 4 classificateurs différents. Nous avons mis en place un programme capable de comparer toutes les combinaisons possibles selon le taux d'erreur et la rapidité d'exécution.

Nous avons aussi réalisé le classificateur All qui reprend le fonctionnement de trois premiers et extrait le label le plus fréquent. Son fonctionnement est simple, mais permet d'obtenir de très bons résultats en dépit de la vitesse. La base de données que nous avons utilisée proposer de nombreuses images très variées et très représentatives des images que pourrait avoir à classer notre programme futur.

Pour finir sur une note personnelle, nous avons aimé travailler sur ce projet et c'est surement l'un des seuls qui aura une utilité après le cours en plus de notre apprentissage sur le sujet.

References

[1] P. Hsu et B.-Y. Chen, 'Blurred image detection and classification,' *Advances in multimedia modeling*, pp. 277–286, 2008.

[2] R. Liu, Z. Li et J. Jia, 'Image partial blur detection and classification,' *Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[3] J. Shi, L. Xu et J. Jia, 'Discriminative blur detection features,' *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2965–2972, 2014.

[4] E. Mavridaki et V. Mezaris, 'No-reference blur assessment in natural images using fourier transform and spatial pyramids,' *Image Processing (ICIP)*, pp. 566–570, 2014.

[5] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis et W. T. Freeman, « Removing camera shake from a single photograph, » *ACM transactions on graphics (TOG)*, pp. 787-794, 2006.

[6] B. Su, S. Lu et C. Tan, 'Blurred image region detection and classification,' *Proceedings of the 19th ACM international conference on Multimedia*, pp. 1397–1400, 2011.

[7] H. Lee et C. Kim, 'Blurred image region detection and segmentation,' *Image Processing (ICIP)*, pp. 4427–4431, 2014.

[8] H. Tong, M. Li, H. Zhang et C. Zhang, 'Blur detection for digital images using wavelet transform,' *Multimedia and Expo*, pp. 17–20, 2004.

[11] 'Understanding Convolutions,' 13 July 2014. [En ligne]. Available: https://colah.github.io/posts/2014-07-Understanding-Convolutions/. [Accès le 9 December 2017].

[9] J. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez et J. Fernandes-Valdivia, « Diatom autofocusing in brightfield microscopy: a comparative study,» *Pattern Recognition. Proceedings. 15th International Conference*, pp. 314–317, 2000.

[10] S. Pertuz, D. Puig et M. A. Garcia, « Analysis of focus measure operators for shape-from-focus, » *Pattern Recognition*, pp. 1415-1432, 2013.

[12] « Documentation Matlab, » [En ligne]. Available: https://www.mathworks.com/help/. [Accès le 09 December 2017].