IUT Paris Descartes

Licence pro (métiers de l'informatique)

M13 – Programmation avancée : analyse d'images

Cahier de TP

Camille Kurtz – camille.kurtz@parisdescartes.fr

2018 – 2019

Prise en main de la librairie de traitement d'images

Dans le cadre de ce module d'initiation au traitement d'images, nous utiliserons une librairie Java open-source nommée PELICAN. Cette librairie, distribuée sous la forme d'une archive .jar peut être directement téléchargée à l'adresse suivante :

`http://www.camille-kurtz.com/teaching/IUT/TI/lib.zip`

PELICAN est une plate-forme logicielle pour l'analyse et le traitement des images, découplant complètement le développement des algorithmes de leur utilisation. Sa prise en main est très simple et de nombreuses fonctionnalités (lecture / enregistrement des images, visualisation des résultats, etc.) sont déjà implémentées.

Installation de la librairie

1. Télécharger le .zip contenant les différentes librairies utiles pour les TPs (dont la librairie PELICAN) à l'adresse énoncée ci-dessus.
2. Après avoir créé un nouveau projet dans Eclipse nommé « M4105C », importer les librairies nécessaires au sein du projet. Pour ce faire :
   (a) Commencer par décompresser l'archive lib.zip à la racine du projet Java (ce projet se trouve dans votre workspace).
   (b) Depuis Eclipse, faire un clique droit sur le projet « M4105C » afin d'accéder à ses propriétés.
   (c) Dans l'option « Java Build Path » (puis onglet « Libraries »), cliquer sur « Add External JARs » puis sélectionner dans le répertoire « lib » du projet les jar des différentes librairies requises (pelican.jar, jfreechart.jar, jai.jar, etc.).
   (d) Valider et les librairies sont maintenant reconnues par votre projet Java.
3. Pour tester la bonne installation de PELICAN, créer une nouvelle classe Java au sein de votre projet comprenant une méthode *main()*. Déclarer une image de la manière suivante `Image test=null;` en prenant soin d'importer dans les headers la classe `Image` de PELICAN via la syntaxe `import fr.unistra.pelican.Image;`. A ce stade votre programme doit compiler normalement (sans croix rouge dans Eclipse).

Instructions de base Dans PELICAN, une image est représentée par la classe abstraite `Image`. Pour charger une image en mémoire, on utilisera la classe `ImageLoader`.

```
//Charger une image en memoire
Image test= ImageLoader.exec("/home/ckurtz/Images/lena.jpg");

//Connaitre la hauteur et la largeur d'une image
int largeur = test.getXDim();
int hauteur = test.getYDim();
```

Pour manipuler les images, vous devrez le plus souvent employer une spécialisation de cette classe qui est la classe `ByteImage` (pour les pixels codés sur 8 bits, soit 256 valeurs). Une fois un objet de la classe `ByteImage` instancié, l'accès aux pixels est réalisé via un système classique de *getValue()* et *setValue(...)*. Voici un exemple illustratif simple pour les images à niveaux de gris (un seul canal) :

```
//Declarer une nouvelle image de taille largeur * hauteur avec 1 canal
ByteImage new_img = new ByteImage(largeur, hauteur, 1, 1, 1);

//Acceder au niveau de gris (la valeur contenue dans le canal 0) du pixel p(5,10)
int r = new_img.getPixelXYBByte(5, 10, 0);

//Affecter la couleur grise [128] au pixel p(100,200)
new_img.setPixelXYBByte(100, 200, 0, 128);
```

et pour les images couleurs (3 canaux RGB) :

```
//Declarer une nouvelle image de taille largeur * hauteur avec 3 canaux RGB
ByteImage new_img = new ByteImage(largeur, hauteur, 1, 1, 3);

//Acceder au canal rouge (canal 0) du pixel p(5,10)
int r = new_img.getPixelXYBByte(5, 10, 0);
//Acceder au canal vert (canal 1) du pixel p(5,10)
int g = new_img.getPixelXYBByte(5, 10, 1);
//Acceder au canal vert (canal 2) du pixel p(5,10)
int b = new_img.getPixelXYBByte(5, 10, 2);

//Affecter la couleur bleu indigo [121, 28, 248] au pixel p(5,10)
new_img.setPixelXYBByte(5, 10, 0, 121);
new_img.setPixelXYBByte(5, 10, 1, 28);
new_img.setPixelXYBByte(5, 10, 2, 248);
```

Pour afficher une image, on utilisera la classe `Viewer2D`.

```
//Afficher une image
new_img.setColor(false); //si false => affichage de chaque canal, si true => affichage d'une image couleur
Viewer2D.exec(new_img);
```

Pour enregistrer une image sur le disque dur, on utilisera la classe `ImageSave`, où `new_img` est l'image que l'on souhaite enregistrer.

```
//Sauvegarder une image
ImageSave.exec(new_img,"/home/ckurtz/Images/maNouvelleImage.jpg");
```

Deux exemples de classes Java (Test1.java et Test2.java) permettant de lire et d'écrire des images sont disponibles à l'adresse suivante :

`http://www.camille-kurtz.com/teaching/IUT/TI/TP01/`

Copier les dans votre projet sous Eclipse.

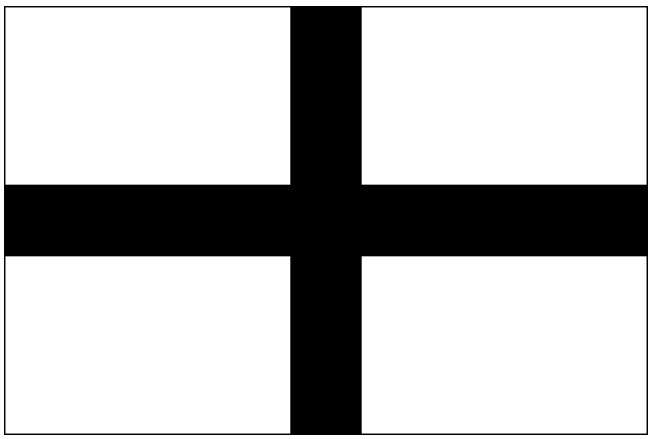

FIGURE 1 – Drapeau maritime de la Bretagne.

1 TP 01 : premiers traitements d'images

1. Des exemples d'images que vous pouvez utiliser pour vos tests sont disponibles à l'adresse suivante (dans le répertoire `img`) : `http://www.camille-kurtz.com/teaching/IUT/TI/TP01/`
2. Deux exemples de classes Java (Test1.java et Test2.java) permettant de lire et d'écrire des images sont disponibles à cette adresse. Copier les dans votre projet sous Eclipse.

Coder les fonctions nécessaires aux traitements ci-dessous :

— **lecture d'une image :** coder une fonction qui prend en entrée une chaîne de caractères (chemin absolu d'une image), qui charge une image (appel à la classe `ImageLoader`) et la retourne en sortie sous la forme d'une variable de type `Image`. Une fois l'image chargée, la fonction devra également affichée la taille (largeur X hauteur) de l'image et le nombre de canaux (1 si image à niveaux de gris et 3 si image en couleur).

— **affichage matricielle d'une image :** coder une fonction qui prend en entrée une image et qui affiche le contenu de cette image sous la forme matricielle. Pour ce faire on va parcourir l'ensemble des pixels de l'image avec 2 boucles imbriquées (une boucle externe qui parcourt les colonnes, les X et une boucle interne qui parcourt les lignes, les Y). Pour chaque valeur de x et y, on affichera la couleur du pixel courant (1 valeur si l'image est à niveaux de gris et 3 valeurs RGB si l'image est en couleur).

— **créer une image synthétique :** coder une fonction sans paramètre d'entrée et qui retourne une image à niveaux de gris du drapeau maritime de la Bretagne. La fonction commencera par construire une nouvelle image vide (largeur = 450, hauteur = 300) à 1 canal. Il faut ensuite affecter la couleur blanche (255) à tous les pixels de l'image en les parcourant. On dessinera ensuite 2 bandes noires (valeur 0) d'une épaisseur de 50 pixels comme illustré dans la figure ci-dessus (il faudra parcourir judicieusement l'image de haut en bas et de gauche à droite ...).

— **extrema d'une image :** coder une fonction qui prend en entrée une image et qui affiche les valeurs minimales et maximales des niveaux de gris des pixels de l'image.

— **inverser une image :** coder une fonction qui prend en entrée une image et qui retourne une inversion symétrique suivant l'axe vertical de cette image. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant cette inversion.

— **inverser aléatoirement :** coder une fonction qui prend en entrée une image et qui retourne une inversion aléatoire des pixels de cette image. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant cette inversion.

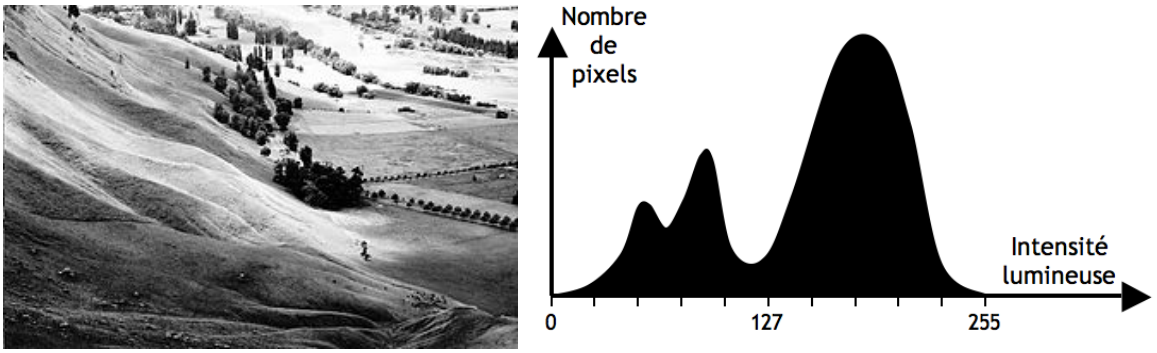

FIGURE 2 – Image et son histogramme colorimétrique.

2 TP 02 : jouons avec les histogrammes et d'autres fonctions

Remarques préliminaires

1. Des exemples d'images que vous pouvez utiliser pour vos tests sont disponibles à l'adresse suivante (dans le répertoire `img`) : `http://www.camille-kurtz.com/teaching/IUT/TI/TP02/`
2. Vous trouverez également à cette adresse une classe Java (HistogramTools.java) permettant d'afficher et d'enregistrer un histogramme passé en paramètre sous la forme d'un tableau de doubles. Copier cette classe dans votre projet sous Eclipse.

Consignes Coder les fonctions nécessaires aux traitements ci-dessous :

— **couleur vers niveau de gris :** coder une fonction qui prend en entrée une image couleur et qui retourne une transformée de cette image en niveaux de gris. Pour un pixel $p(x,y)$ de couleur (r,g,b), lui affecter le niveau de gris v où $v = (r+g+b)/3$. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.

— **seuillage / binarisation :** coder une fonction qui prend en entrée une image à niveaux de gris, et un seuil S et qui retourne une image binaire. Pour un pixel $p(x,y)$ de niveau de gris v, lui affecter le niveau de gris 0 si $v \leq S$ ou 255 sinon. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.

— **étirement de contraste :** Supposons que dans une image donnée, l'intervalle des niveaux de gris utilisés ne s'étende pas de 0 à 255. Soient g_{min} et g_{max} les niveaux de gris minimum et maximum des pixels de l'image. L'étirement de contraste consiste en l'application aux niveaux de gris v de l'image d'une fonction f telle que : $f(v) = 255(v - g_{min})/(g_{max} - g_{min})$. Coder une fonction qui prend en entrée une image à niveaux de gris et qui retourne une image dont le contraste est étiré. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image.

— **histogramme d'une image :** coder une fonction qui prend en entrée une image à niveaux de gris et qui retourne son histogramme. Un histogramme peut être représenté par un tableau de taille 256 où chaque case $i \in [0,255]$ représente le nombre de pixels dans l'image ayant le niveau de gris (i). Pour ce faire on va commencer par initialiser un tel tableau avec toutes les cases à O. Parcourir ensuite l'ensemble des pixels de l'image avec 2 boucles imbriquées (une boucle externe qui parcourt les colonnes, les X et une boucle interne qui parcourt les lignes, les Y). Pour chaque valeur de x et y, on récupérera la valeur de niveau de gris v du pixel p_i et on incrémentera de 1 la case correspondante dans le tableau. Utiliser ensuite une des méthodes de la classe (HistogramTools.java) pour afficher cet histogramme.

— **égalisation d'histogramme :** coder une fonction qui prend en entrée une image à niveaux de gris, qui calcule son histogramme puis qui l'égalise (voir le support de CM 02).

— **diminuer/agrandir par 2 la taille d'une image :** coder une fonction qui prend en entrée une image et qui retourne une image 2 fois **moins grande** que l'image d'origine ; chaque pixel $p'(x, y)$ de la nouvelle image sera représenté comme la moyenne de 4 pixels voisins ($p(x, y)$, $p(x+1, y+1)$, $p(x+1, y)$, et $p(x, y+1)$) dans l'ancienne image. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image de taille (largeur/2 * hauteur/2). Coder également une fonction qui prend en entrée une image et qui retourne une image 2 fois **plus grande** que l'image d'origine.

FIGURE 3 – Résultat du filtre médian sur une image en couleurs RGB bruitée.

3 TP 03 : jouons avec les filtres

Remarques préliminaires

1. Des exemples d'images que vous pouvez utiliser pour vos tests sont disponibles à l'adresse suivante (dans le répertoire `img`) : `http://www.camille-kurtz.com/teaching/IUT/TI/TP03/`
2. Vous trouverez également à cette adresse une classe Java (NoiseTools.java) permettant d'insérer artificiellement du bruit dans une image passée en paramètre. Copier cette classe dans votre projet sous Eclipse.

Consignes Coder les fonctions nécessaires aux traitements ci-dessous :

— **filtre moyenneur sur image à niveaux de gris :** coder une fonction qui prend en entrée une image à niveaux de gris et qui retourne une transformée de cette image en niveaux de gris. Remplacer le niveau de gris d'un pixel $p(x,y)$ par la moyenne des niveaux de gris des 8 pixels voisins ($p(x,y)$, $p(x+1,y)$, $p(x+1,y+1)$, $p(x-1,y-1)$, $p(x-1,y)$, $p(x,y-1)$, $p(x-1,y+1)$, $p(x+1,y-1)$, et $p(x,y+1)$). La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.

— **filtre médian sur image à niveaux de gris :** coder une fonction qui prend en entrée une image à niveaux de gris et qui retourne une transformée de cette image en niveaux de gris. Remplacer le niveau de gris d'un pixel $p(x,y)$ par la valeur médiane des niveaux de gris des 8 pixels voisins ($p(x,y)$, $p(x+1,y)$, $p(x+1,y+1)$, $p(x-1,y-1)$, $p(x-1,y)$, $p(x,y-1)$, $p(x-1,y+1)$, $p(x+1,y-1)$, et $p(x,y+1)$). La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée. Comparer l'effet du filtre moyenneur et du filtre médian sur l'image de Lenna bruitée.

— **passons à la couleur :** reprendre les 2 méthodes précédentes et les étendre pour permettre des traitements sur des images en couleur. On travaillera de manière indépendante sur chaque canal RGB.

— **régularisation d'une image à niveaux de gris :** coder une fonction qui prend en entrée une image à niveaux de gris et qui retourne une transformée régularisée de cette image en niveaux de gris. Une telle transformation repose sur l'application d'un filtre de convolution et permet de diminuer le flou dans une image (voir support de cours CM 03). La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.

— **détecteur de contours sur image couleur :** coder une fonction qui prend en entrée une image en couleurs et qui retourne une transformée de cette image en niveaux de gris. Vous utiliserez le filtre de Sobel pour extraire les contours de l'image (voir support de cours CM 03). Pour un pixel donné, on travaillera de manière indépendante sur chaque canal RGB puis on sommera le résultat obtenu pour obtenir une unique valeur qu'on tronquera au-delà de 255. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.

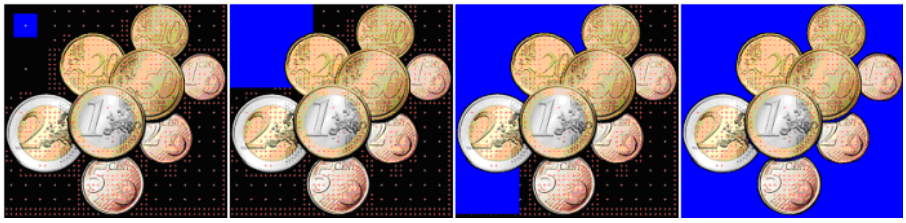

FIGURE 4 – Croissance progressive d'une région.

4 TP 04 : segmentons une image

Remarques préliminaires

1. Des exemples d'images que vous pouvez utiliser pour vos tests sont disponibles à l'adresse suivante (dans le répertoire `img`) : `http://www.camille-kurtz.com/teaching/IUT/TI/TP04/`
2. Vous trouverez également à cette adresse une classe Java (SegmentationTools.java) permettant d'afficher les frontières des régions issues d'une segmentation Copier cette classe dans votre projet sous Eclipse. Observer l'appel aux classes *LabelsToRandomColors* et *FrontiersFromSegmentation*.

Consignes Coder les fonctions nécessaires aux traitements ci-dessous :

— **segmentation par seuillage d'Otsu sur image à niveaux de gris :** Le seuillage est une opération qui consiste à binariser une image, c'est-à-dire la transformer en une image dont les pixels sont noirs ou blancs (cf TP 2). Il s'agit d'une méthode simple de segmentation : les pixels de l'image sont séparés en deux classes distinctes. Rappel : on peut l'implémenter en codant une fonction qui prend en entrée une image à niveaux de gris, et un seuil S et qui retourne une image binaire. Pour un pixel $p(x,y)$ de niveau de gris v, lui affecter le niveau de gris 0 si $v \leq S$ ou 255 sinon.
Souvent, le seuillage est utilisé pour segmenter un objet d'intérêt du fond de l'image. Cependant, la valeur de seuil S doit être fixée manuellement par l'utilisateur, ce qui n'est pas toujours souhaitable pour effectuer un traitement automatique.
La **méthode d'Otsu** est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image. L'algorithme suppose alors que l'image à binariser ne contient que deux classes (groupes) de pixels, (c'est-à-dire le premier plan et l'arrière-plan) puis calcule le seuil optimal qui sépare ces deux classes afin que leur variance intra-classe soit minimale (les classes sont les plus compactes possible). D'un point de vue algorithmique, il est plus pratique de chercher à maximiser la variance inter-classe (les classes sont les plus éloignées possible) via la formule suivante :

$$\sigma^2_{inter}(S) = \omega_1(S)\omega_2(S)[\mu_1(S) - \mu_2(S)]^2 \quad (1)$$

où $\omega_1(S)$ et $\omega_2(S)$ désignent les probabilités qu'un pixel soit dans l'une ou l'autre classe. Les valeurs sont toutes calculables facilement à partir de l'histogramme de l'image.
Coder une fonction qui prend en entrée une image à niveaux de gris, et un seuil S et qui retourne une image binaire seuillée par la méthode d'Otsu. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.
Algorithme

1. Calculer l'histogramme et les probabilités de chaque niveau d'intensité
2. Définir les $\omega_i(0)$ et $\mu_i(0)$ initiaux

3. Parcourir tous les seuils possibles $S = 1 \ldots$intensité max
   (a) Mettre à jour ω_i et μ_i
   (b) Calculer $\sigma^2_{inter}(S)$
4. Le seuil désiré correspond au $\sigma^2_{inter}(S)$ maximum.

Plus de détails sur : `https://en.wikipedia.org/wiki/Otsu%27s_method`

— **segmentation par croissance de régions :** L'approche proposée pour segmenter une image consiste à faire croître une région autour d'un pixel de départ $p_(x, y)$ nommée graine. L'agglomération des pixels n'exploite aucune connaissance a priori de l'image. En fait, la décision d'intégrer à la région un pixel voisin repose seulement sur un critère d'homogénéité imposé à la zone en croissance.
Algorithme
– Créer la liste « [S] » des points de départs (cette liste peut être réduite à un point)
– Pour chaque pixel « P » dans la liste « [S] »

1. Si le pixel « P » est déjà associé à une région, alors prendre le pixel « P » suivant dans la liste « [S] »
2. Créer une nouvelle région « [R] » (par exemple un *TreeSet* de *Points*)
3. Ajouter le pixel « P » dans la région « [R] »
4. Calculer la valeur/couleur moyenne de « [R] »
5. Créer la liste « [N] » des pixels voisins du pixel « P »
6. Pour chaque pixel « Pn » dans la liste « [N] »
   (a) Si (« Pn » n'est pas associé à une région ET « R + Pn » est homogène) Alors
   (b) Ajouter le pixel « Pn » dans la région « [R] »
   (c) Ajouter les pixels voisins de « Pn » dans la liste « [N] »
   (d) Recalculer la valeur/couleur moyenne de « [R] »
   (e) Fin Si
7. Fin Pour

–Fin Pour

L'indicateur d'homogénéité peut être construit à partir d'une mesure de similarité :
Indicateur : "vrai" si $Homogénéite(R) <= SEUIL$, "faux" sinon.

En utilisant la définition de la variance et la formule du calcul de la distance entre 2 couleurs, on obtient : Homogénéite(R) = Variance(pixels de R) = Moyenne(distance(pixels de R, Moyenne(pixels de R))^2)
Plus de détails sur :
`https://xphilipp.developpez.com/articles/segmentation/regions/?page=page_2`

— **segmentation par décomposition/fusion (split/merge) :**
Plus de détails sur :
`https://xphilipp.developpez.com/articles/segmentation/regions/?page=page_3`

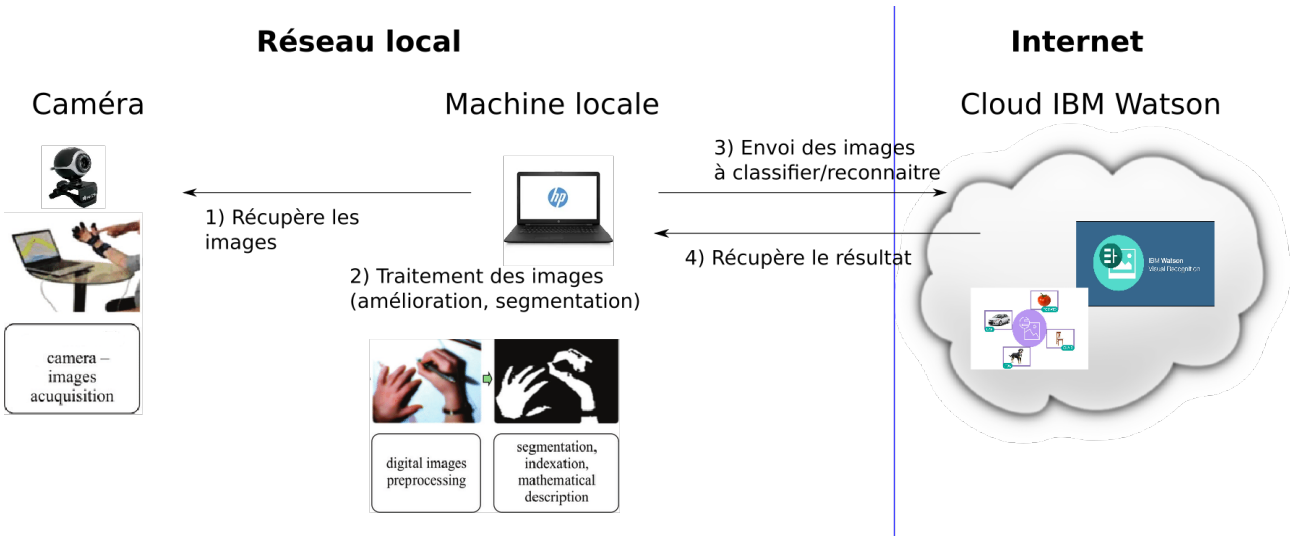

FIGURE 5 – Architecture considérée.

5 TP 05 : reconnaissance d'objets à partir d'une caméra

Remarques préliminaires

1. Dans le cadre de ce TP, nous voulons mettre en place un système permettant de reconnaître des objets ou des personnes dans des images ayant été acquises par une caméra. La figure ci-dessus illustre les différents modules à mettre en place pour construire un tel système.
2. Vous trouverez à cette adresse des classes Java permettant d'effectuer certains traitements relatifs à l'architecture que nous souhaitons mettre en place. Copier ces classes dans votre projet sous Eclipse. `http://www.camille-kurtz.com/teaching/IUT/TI/TP05/`

Consignes Coder les fonctions nécessaires aux traitements ci-dessous :

— **1) Récupérer les images d'une caméra :** Une caméra produit un flux d'images en continu. L'objectif est ici de récupérer une image du flux à un instant donné. La caméra peut être distante (caméra IP) sur le réseau local ou directement intégrée dans la machine locale effectuant le traitement. Pour manipuler le flux de la caméra et récupérer les images, nous allons utiliser la librairie Java suivante : `http://webcam-capture.sarxos.pl/` La classe (Camera.java) illustre comment récupérer une image compatible au format Pelican.

— **2) Traitement des images :** Une fois l'image récupérée de la caméra, il convient d'effectuer certains pré-traitements afin d'opérer la reconnaissance des objets/personnes présentes dans l'image. Par exemple, il peut être nécessaire d'améliorer le contraste de l'image, de rendre son contenu moins flou ou encore de segmenter les objets présents dans cette dernière avant de les envoyer au système de reconnaissance. Dans le cas où vous souhaiteriez utiliser ce système pour reconnaître un objet particulier et si l'objet a été photographié sur un fond clair, vous pouvez employer un algorithme de segmentation (region growing ou binarisation Otsu) pour supprimer le fond de l'image et ainsi créer une nouvelle image où seul l'objet d'intérêt est présent.

— **3) Classification/étiquetage des objets :** Une fois l'image pré-traitée, la prochaine étape est de classifier/étiqueter les objets d'intérêt présents dans son contenu. Nous allons ici utiliser un algorithme de classification reposant sur des réseaux profonds (deep learning) capable de prédire une étiquette pour une image donnée. Pour ce faire, nous allons faire appel au

webservice *IBM Watson Visual Recognition* présent sur le cloud IBM. Watson est un programme informatique d'intelligence artificielle conçu par IBM. Le service Visual Recognition permet de reconnaître au sein d'une image donnée en paramètre des objets ou des personnes. On peut envoyer au serveur une image et ce dernier va renvoyer un fichier json contenant la liste des objets reconnus dans l'image. Commencer par vous rendre à l'adresse suivante pour tester la démo :

`https://www.ibm.com/watson/services/visual-recognition/`

Pour pouvoir interroger ce webservice à distance (sous la forme de requête HTTP POST ou GET) il vous faut créer un compte de développeur pour bénéficier d'une clé API à fournir lors de vos requêtes au webservice :

`https://console.bluemix.net/docs/services/visual-recognition/getting-started.html#getting-started-t`

Une fois la clé API obtenue, nous pouvons utiliser la classe (VisualRecognitionExample.java) qui permet d'envoyer une image au serveur IBM. A noter qu'il est également possible d'utiliser un webservice spécifique pour la reconnaissance de visages.

— **4) Exploitation des résultats :** Le serveur retourne un résultat au format JSON qui contient la liste des objets reconnus. Vous êtes libres de choisir la manière dont vous pouvez exploiter ces résultats (les stocker dans un fichier texte, une BD, sous la forme d'une page web, envoyer un email automatique pour prévenir une personne, etc.).

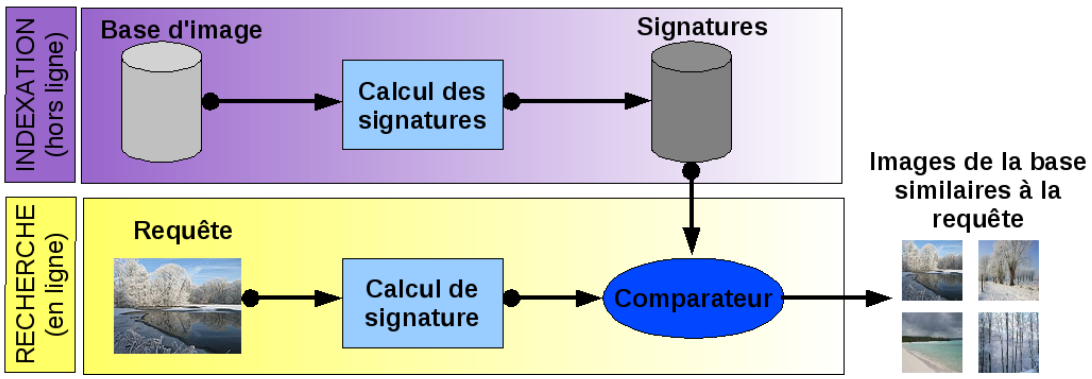

FIGURE 6 – Recherche d'images par le contenu (en anglais : Content Based Image Retrieval).

6 Sujet de projet : recherche d'images par le contenu

La recherche d'images par le contenu (en anglais : Content Based Image Retrieval ou CBIR) est une technique permettant de rechercher des images à partir de ses caractéristiques visuelles, c'est-à-dire induite de leurs pixels (par exemple l'histogramme de l'image). Un cas typique d'utilisation est la recherche par l'exemple où l'on souhaite retrouver des images visuellement similaires à un exemple donné en requête. Il s'oppose à la recherche d'images par mots clés ou tags, qui fut historiquement proposée par les moteurs de recherche tels que Google Image grâce à des banques d'images où les images sont retrouvées en utilisant le texte qui les accompagne plutôt que le contenu de l'image elle-même (Google Image propose désormais des filtres basés sur le contenu (pixels) des images).

Cette technologie s'est développée dans les années 90 pour la recherche de données dans les secteurs industriels, l'imagerie médicale ou cartographiques. Elle a donné lieu à de nombreux programmes et produits de la recherche. La reconnaissance faciale est utilisée par exemple par Interpol ou Europol pour rechercher les criminels. L'application la plus mature à la fin des années 2000 est la recherche de copie, utilisée dans la lutte contre la contrefaçon.

6.1 Principe

Le principe général de la recherche d'images par le contenu comporte deux étapes (voir la figure 6). Lors d'une première phase (**étape d'indexation**), on calcule les signatures des images et on les stocke dans une base de donnée. La seconde phase, dite de **étape de recherche** se déroule de la manière suivante. L'utilisateur soumet une image comme requête. Le système calcule la signature selon le même mode que lors de la première phase d'indexation. Ainsi, cette signature est comparée à l'ensemble des signatures préalablement stockées pour en ramener les images les plus semblables à la requête.

Lors de la phase d'indexation, le calcul de signature consiste en l'extraction de caractéristiques visuelles des images telles que : la texture, les formes, la couleur, etc. Dans le cadre de ce projet, nous utiliserons ici comme signature l'histogramme couleur d'une image.

Une fois ces caractéristiques extraites, la comparaison consiste généralement à définir une mesure de similarité entre deux images (nommée « comparateur » dans la figure 6). Au moyen de cette mesure de similarité et d'une image requête, on peut alors calculer l'ensemble des valeurs de similarité entre cette image requête et l'ensemble des images de la base d'images. Il est ensuite possible d'ordonner les images de la base suivant leur score, et présenter le résultat à l'utilisateur, les images de plus grand score étant considérées comme les plus similaires.

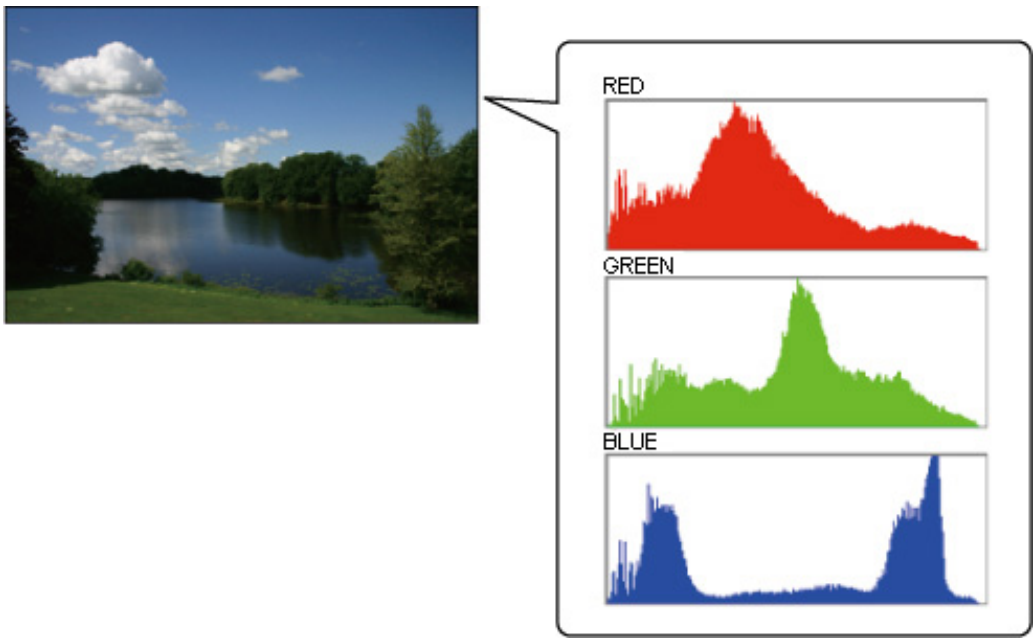

FIGURE 7 – Histogramme couleur d'une image.

6.2 Déroulement du projet

L'objectif du projet est d'implémenter un prototype permettant à un utilisateur d'effectuer une recherche d'images par le contenu. Cette implémentation Java sera réalisée à l'aide de la librairie PELICAN et des fonctions déjà codées durant les 3 premières séances de TP.

Partie 1 : premier prototype A partir d'une image requête fournie en entrée par l'utilisateur, le système devra rendre en sortie (sous la forme d'un affichage) les n images de la base de données les plus similaires à l'image requête, triées par ordre de similarité. Nous nous focaliserons ici uniquement sur les images en couleur. Afin de simplifier le problème, la signature d'une image sera ici modélisée par l'histogramme colorimétrique de l'image et nous n'implémenterons pas, dans un premier temps, le système d'index (les signatures seront calculées *à la volée* pour chaque nouvelle requête). Deux bases d'images que vous pouvez utiliser pour vos tests sont disponibles à l'adresse suivante :

`http://www.camille-kurtz.com/teaching/IUT/TI/Projet/images/`

Vous êtes libres également d'utiliser d'autres bases d'images ou de compléter ces dernières. Voici les étapes principales du prototype à réaliser :

1. **Lecture en mémoire d'une image requête** R. Appliquer un filtre médian sur cette image afin de potentiellement la dé-bruiter ; Afficher cette image avec le `Viewer2D` ;
2. **Construction de l'histogramme couleur** H_R de l'image requête : ⚠ on considère ici une image en couleur RGB (et non à niveaux de gris), il faut donc modifier la fonction de calcul d'histogramme implémentée précédemment en TP pour permettre la construction d'un histogramme couleur. Un histogramme couleur sera représenté par une matrice $H[256][3]$ contenant 3 lignes (une pour chaque canal R, G et B) et 256 colonnes (voir Figure 7). Un histogramme couleur est donc un histogramme à 3 dimensions.
3. **Discrétisation de l'histogramme** : Un histogramme classique est composée de 256 barres par canal (une barre par niveau de gris possible). Cette dimension est élevée et peut causer

des problèmes dans la fonction de comparaison lors de la recherche d'images similaires (non prise en compte de la corrélation entre des valeurs proches d'intensité). Pour pallier à ce problème, une stratégie consiste à réduire le nombre de barres de l'histogramme en discrétisant (diminuant) l'espace des valeurs. Coder une fonction, prenant en paramètre un histogramme $H[256][3]$, permettant de discrétiser un histogramme en divisant par 10 le nombre de barres de l'histogramme : la première barre codera les 25 premiers niveaux d'intensité, la deuxième barre codera les 25 niveaux suivants, etc.

4. **Normalisation de l'histogramme** : Pour que 2 histogrammes issues de 2 images de tailles différentes soient comparables, il est nécessaire de les normaliser par rapport au nombre de pixels dans l'image. Coder une fonction, prenant en paramètres un histogramme ainsi que le nombre de pixels dans une image, permettant de normaliser un histogramme en divisant chaque barre de l'histogramme par le nombre de pixels dans l'image afin d'obtenir une valeur de pourcentage.
5. **Recherche des images similaires** : parcourir à l'aide d'une boucle le dossier stockant la base d'images (en prenant soin d'ignorer l'image requête R si cette dernière est dans le dossier) :
   (a) Pour chaque image I de la base, appliquer un filtre médian sur cette image afin de potentiellement la dé-bruiter, puis construire l'histogramme couleur H_I de l'image (le discrétiser et le normaliser) ;
   (b) Calculer la similarité entre les 2 images R et I en mesurant la distance de similarité entre H_R et H_I (voir aide ci-dessous) ;
   (c) Stocker dans une structure de données maintenue triée (`TreeMap` par exemple ...) le nom de l'image I ainsi que sa valeur de similarité avec l'image requête R.
6. Afficher par odre de similarité les 10 premières images les plus similaires à l'image requête R (en égalisant au préalable leurs histogrammes). Attention la distance Euclidienne utilisée ici est une mesure de dissimilarité : les images de plus grand score sont considérées comme les plus dissimilaires.

Mesure de la similarité entre histogrammes
Dans de nombreuses applications, il est utile de calculer une distance (ou plus généralement une mesure de similarité) entre histogrammes. On définit pour cela une mesure pour calculer si deux histogrammes sont « proches » (se ressemblent) l'un de l'autre. Soit H_1 et H_2 deux histogrammes de même taille N (i.e. avec le même nombre de barres). La distance la plus utilisée est la distance Euclidienne :

$$d_{Eucl}(H_1, H_2) = \sqrt{\sum_{i=1}^{N} (H_1^i - H_2^i)^2}$$

où H_1^i représente la i-ème barre de l'histogramme H_1. C'est exactement le même principe que quand vous avez 2 points $p_1(x_1, y_1)$ et $p_2(x_2, y_2)$ dans le plan Euclidien $\mathbb{R}^2$ et que vous voulez connaître la distance entre ces 2 points via la formule $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Plus la valeur de la distance entre 2 histogrammes sera grande moins ces histogrammes seront similaires et inversement. La distance entre un histogramme et lui-même est nulle.
Pour mesurer la similarité entre 2 histogrammes couleur à 3 dimensions, on calcule indépendamment les distances euclidiennes entre les histogrammes représentant le canal R, G et B, puis on somme ces 3 valeurs de distance.

Partie 2 : utilisation d'un système d'indexation Afin d'éviter de recalculer à chaque fois (pour chaque nouvelle requête) l'ensemble des signatures (des histogrammes) des images du jeu de données, on pourra implémenter un système d'indexation basique. Pour ce faire, vous devrez rajouter au prototype déjà implémenté un pré-traitement qui parcourra chaque image de la base et qui

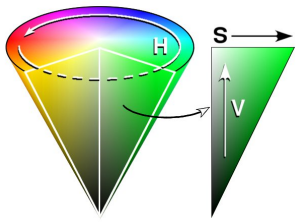

FIGURE 8 – Espace de couleur HSV (Hue-Saturation-Value).

calculera son histogramme couleur (normalisé et discrétisé). Ces histogrammes devront alors être stockés en mémoire physique (sous la forme d'un fichier texte ou dans une base de données SQ-Lite par exemple). Pendant l'étape de recherche d'images similaires, il ne sera donc plus nécessaire de calculer l'histogramme de chaque image de la base d'images mais simplement de récupérer en mémoire (ou en base de données) son histogramme, diminuant ainsi les temps de calculs lors de la recherche.

Partie 3 : utilisation d'un autre espace couleur La couleur est le descripteur visuel le plus employé, car c'est le plus perceptuel. L'espace RGB est très simple à utiliser, car c'est celui employé par de nombreux appareils de capture d'images qui effectuent leurs échanges d'informations uniquement en utilisant les triplets (R,G,B). Cependant, ces trois composantes sont fortement corrélées (par exemple, si l'on diminue la composante verte, la teinte parait plus rouge), de plus l'espace RGB est sensible aux changements d'illumination, et ne correspond pas au processus de perception humaine. L'espace HSV (Hue-Saturation-Value, voir la Figure 8) sépare les informations relatives à la teinte (Hue), la saturation (Saturation) et l'intensité (Value). Cet espace est plus intuitif à utiliser car il correspond à la façon dont nous percevons les couleurs. La teinte décrit la couleur (rouge, vert ...), la saturation décrit l'intensité de la couleur, et la valeur décrit la luminosité de la couleur.

Le travail demandé consiste ici à étendre le prototype pour permettre de retrouver des images similaires en se basant sur une signature d'image, toujours à base d'histogramme, mais dans l'espace HSV. Reprendre les étapes précédentes (Partie 1), où les images seront maintenant décrites par un histogramme couleur H_R calculé en considérant l'image dans l'espace HSV. Attention dans cet espace, les composantes V et S ne varient plus entre 0 et 255 mais entre 0.0 et 1.0. Par ailleurs, la composante H est exprimée en degrés et varie entre 0 et 360.

Pour la conversion d'un pixel de l'espace RGB à l'espace HSV, voir le lien ci-dessous :

`http://www.had2know.com/technology/hsv-rgb-conversion-formula-calculator.html`

Partie 4 : évaluation de la qualité du système Vous disposez maintenant de 2 systèmes différents permettant de retrouver (et d'ordonner), à partir d'une image requête, les images les plus semblables dans une base d'images. Pour être en mesure d'évaluer la qualité et les performances de votre système, il est nécessaire d'évaluer les résultats obtenus par ce dernier qui correspondent aux images retrouvées. Pour ce faire, la première étape consiste à regrouper l'ensemble des images de la base en différentes catégories, suivant le contenu de ces dernières (paysages de montagne, photos de plages, etc.). Ensuite, étant donnée une nouvelle image requête R (par exemple un paysage de montagne), et un ensemble d'images résultats retrouvées par le système, on peut évaluer la qualité de ces résultats en comptant le nombre d'images retrouvées (parmi les 10 premières) qui appartiennent à la catégorie *paysages de montagne*. Ce chiffre correspond à la précision du système pour la requête R et peut être exprimé sous la forme d'un pourcentage. En considérant

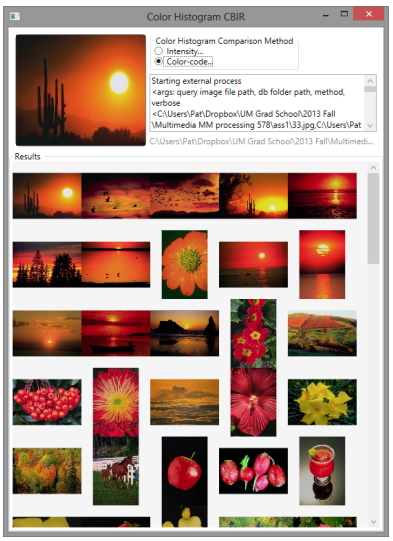

FIGURE 9 – Exemple d'une interface graphique permettant de visualiser l'image requête et les images similaires retrouvées dans la base.

chaque image de la catégorie *paysages de montagne* comme une nouvelle requête et en calculant la précision pour chacune d'elle, on peut alors obtenir la précision moyenne du système pour la catégorie *paysages de montagne*. D'autres mesures sont possibles

Le travail demandé consiste ici à :

- Créer un ensemble de catégories d'images (paysages de montagne, photos de plages, etc.) à partir de toutes les images de la base ;
- Développer un système permettant de calculer pour chaque catégorie un score de précision moyen ;
- Comparer les résultats obtenus avec votre prototype de CBIR dans l'espace RGB et celui dans l'espace HSV ; quelles catégories ont été les mieux reconnues ?
- Comparer les résultats obtenus avec votre meilleur prototype et celui des autres binômes (quel le meilleur gagne !) ;
- Evaluer vos résultats avec d'autres mesures comme la R-Précision, la Mean Precision et la Mean average precision (voir `https://en.wikipedia.org/wiki/Information_retrieval`).

Partie 5 : interface graphique Une fois que toutes les étapes précédentes ont été validées, vous pouvez réaliser une interface graphique de base (en Java ou en générant dynamiquement une page Web html) permettant de visualiser l'image requête et les images similaires retrouvées dans la base. La figure 9 présente un exemple d'interface graphique.

6.3 Consignes

- Projet à faire en binôme **(pas de trinôme)**.
- Évaluation et présentation en dernière semaine de TD. A l'issue de la présentation, le code source (.zip) devra être envoyé par email à votre enseignant de TP.
- Pour la présentation, chaque étudiant(e) devra apporter une version compilée du projet (qui fonctionne sur les machines de l'IUT) et réaliser une démonstration du programme devant un encadrant de TP.
- Toute absence injustifiée entraînera une note nulle au projet.
- **Afin de détecter une triche potentielle, les projets seront comparés via un outil automatique de détection de fraudes.**