Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet libre DI5
2015-2016

Méthode de dégradation image par re-compression JPEG et re-échantillonnage

Tuteurs académiques
Mathieu Delalandre
Sebastien Aupetit

Étudiants
ZHANG Shimeng (DI5)

25 septembre 2016

Liste des intervenants

Nom	Mail	Qualité
ZHANG Shimeng	shimeng.zhang@etu.univ-tours.fr	Étudiant DI5
Mathieu Delalandre	mathieu.delalandre@univ-tours.fr	Tuteur académique, Département informatique
Sebastien Aupetit	sebastien.aupetit@univ-tours.fr	Tuteur académique, Département informatique

Avertissement

Ce document a été rédigé par ZHANG Shimeng susnommé les auteurs.

L'école polytechnique de l'université François Rabelais de Tours est représentée par Mathieu Delalandre et Sebastien Aupetit susnommé les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respects des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document :
ZHANG Shimeng, *Méthode de dégradation image par re-compression JPEG et re-échantillonnage*, Projet libre DI5, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={Shimeng, ZHANG},
    title={Méthode de dégradation image par re-compression JPEG et re-échantillonnage},
    type={Projet libre DI5},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2015/11/18)

Table des matières

Table des figures

1 Présentation et objectif du projet

1 Problématique : Détection de duplicata

La problématique de ce projet est la détection de duplicata. Une image et son duplicata représentent visuellement un même contenu, mais le codage des images est différent due aux modes de capture, bruit, modes de compression, contraste, formatages des images, etc.

Dans ce projet, on s'intéressera à fabriquer les duplicatas des images de référence par une dégradation des images sur deux niveaux :

- Recompression
- Sous-échantillonnage

2 Base de données Manga109 et ImageJ

Manga109 est une base de données libre de droit à des fins d'usage de recherches en traitement d'images et reconnaissance de forme.

Elle se compose de 109 magazines Manga et 200 000 pages environ qui ont les même dimensions de 827 x 1170. Il est aussi important de noter que la base de donnée Manga109 ne contient que les images de format .jpg(JPEG).

ImageJ est un logiciel open source qui est développé en Java. Il est beaucoup utilisé dans le domaine de traitement et d'analyse d'images, parce qu'il permet de faire les analyses de base et aussi d'ajouter des nouvelles fonctionnalités via les plugins et macros.

Comme toutes les images dans Manga109 sont en niveaux de gris, ImageJ permet de traiter les images de :

- 8-bit : entiers positifs sur 8 bits
- 16-bit : entiers positifs sur 16 bits
- 32-bit : flottants (réels) signés sur 32 bits
- 8-bit color et RGB color
- RGB color et HSB color

JPEG est une norme qui définit le format d'enregistrement et de décodage pour une représentation numérique compressée d'une image fixe.

Dans Manga109, il n'y a que les images de format .jpg. Si on les ouvres dans imageJ, le type par défaut est de 8-bit.

2 Analyse des mécanismes de traitement d'image

1 Modifier une image JPEG par recompression

Au sein de ce projet, on va faire les duplicatas des images par recompression en modifiant le facteur de qualité.

Compression JPEG

J'aimerais vous parler de la compression JEPG avant de rentrer dans le détail. La compression JPEG c'est (cf : Figure 1) :

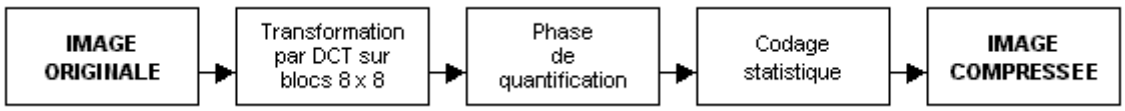

Figure 1 – *Compression JPEG*

- un travail sur des blocs (8x8)
- une utilisation d'une DCT sur un bloc
- une quantification par une matrice de quantification
- un codage en général avec perte

Il est important de savoir qu'il y a beaucoup de méthodes pour faire une compression d'image. En général, il y a deux types de compression : avec perte et sans perte. L'algorithme de compression JPEG est beaucoup utilisé pour les compressions avec perte avec un mécanisme de calculer DCT [WWW2].

Le DCT se traduit par Discrete Cosine Transform. La compression JPEG repose sur une transformation en DCT locale sur des carrés 8 par 8. Autrement dit, on doit découper l'image de départ par des blocs de 8x8 qui contiennent les fréquences différentes de cette image[WWW5].

Dans l'étape suivante, un mécanisme de qualification sera appliqué sur les blocs de DCT. Il est important de savoir que la vraie compression commence par ici en supprimant les fréquences moins importantes, ce qui signifie le type "avec perte".

Comme la base de données Manga109 ne contient que les images en gris, la procédure de compression est :

- D'abord, l'image de départ sera découpée en bloc 8x8 pixels
- La formule DCT sera appliquée sur chaque bloc de gauche à droite et du haut vers le bas (cf : Figure 2).
- Ensuite, chaque bloc sera compressé par une qualification

- Parcourir les blocs compressés avec une lecture zigzag
- A la fin, si on a besoin de reconstruire cette image, l'IDCT (Inverse Discrete Cosine Transform) sera calculé pour une décompression d'image de départ

Pour être plus précis, je vais vous expliquer étape par étape (cf : Figure 1), mais la perte de qualité se situe principalement pendant la quantification [WWW2].

Étape 1 : Formule DCT et Matrice DCT

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases}$$

Figure 2 – *Formule DCT*

Comme on travail sur un découpage de 8x8, le N = 8. La matrice p présente les valeurs de chaque pixel sur chaque bloc de l'image de départ. Ensuite , on applique le calcul de DCT sur chaque bloc avec une matrice de transformation T.

Étape 2 : Qualification et facteur de qualité JPEG

L'étape de quantification est la principalz de la dégradation de l'image. C'est lors de cette étape que l'image va être réellement compressée.

La qualification sur la matrice DCT est théoriquement inversible. L'essentiel de la compression est réalisée en quantifiant les coefficients de la DCT, c'est à dire en les arrondissant.

Dans la compression JPEG, chaque DCT(x,y) est divisé par une constante q(u,v), le pas de quantification du niveau DCT correspondant. La table des q(u,v) est appelée la matrice de quantification qui contient un facteur de qualité(cf : Figure 3).

$$C_{i,j} = round\left(\frac{D_{i,j}}{Q_{i,j}}\right)$$

Figure 3 – *Qualification*

Le facteur de qualité JPEG n'a pas une définition standard. Mais le principe en reste principalement basé sur la quantification plus ou moins « sévère » des coefficients DCT[3].

Il est important de savoir que si le facteur de qualité JPEG est fixé à 1, il fournit une compression maximale (donc une qualité minimale). Sa limite de 100, correspondant à une compression minimale (donc une qualité maximale).

Étape 3 : Codage JPEG

Après la qualification, il est normal qu'il contient beaucoup de 0 dans la matrice obtenue. En suite on utilise une lecture de Zigzag pour rassembler les faibles niveaux DCT(cf : Figure 4).

Ensuite, la matrice 1x64 sera compressée par un codage statistique tel que le codage de Huffman. Sinon on peut aussi considérer le codage JPEG comme un filtrage de l'image (DCT et quantification) suivi d'un codage entropique qui propose une compression sans perte : Lecture zigzag + DPCM ou RLE (run-length encoding) + Huffman[4][WWW2].

Étape 4 : Décompression

Jusqu'ici, l'image départ est compressée. Si on a besoin de reconstruire l'image, on peut appliquer la méthode de décompression. C'est à dire que l'image compressée sera décompressée par la méthode de décodage statistique, puis la matrice obtenue sera multipliée par la matrice de quantification que l'on reconstituera grâce au facteur de qualité et enfin on appliquera la DCT inverse pour retrouver une image plus ou moins dégradée par rapport à l'image de départ[WWW1].

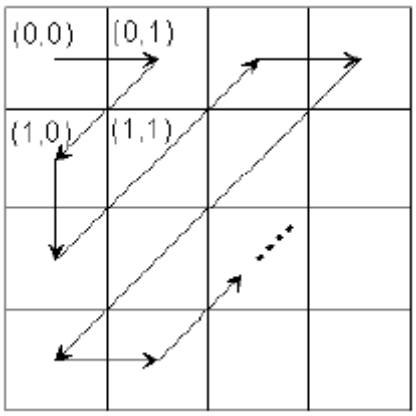

Figure 4 – *Lecture Zigzag*

2 Une méthode de calcul de bruit : PSNR

PSNR (Peak Signal to Noise Ratio) est une mesure de distorsion utilisée en image numérique. Elle est considérée comme une bonne mesure des bruits aléatoires.

Comme ce projet consiste sur la recompression d'images, le PSNR peut aider à mesurer :

- La performance des codages
- La qualité de reconstruction de l'image compressée par rapport à l'image originale

Le PSNR est calculé par (cf : Figure 5) :

$$PSNR = 10 \cdot \log_{10}\left(\frac{d^2}{EQM}\right)$$

Figure 5 – *PSNR*

où d est la valeur maximum possible pour un pixel. Dans le cas standard d'une image où les composantes d'un pixel sont codées sur 8 bits, d = 255.

EQM (Erreur Quadratique Moyenne) est calculé pour 2 images I_o et I_r de la même taille n*m comme (cf : Figure 6) :

$$EQM = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(I_o(i,j) - I_r(i,j))^2$$

Figure 6 – *EQM*

Il est aussi important de noter que les valeurs de PSNR s'expriment en dB, plus il est élevé et meilleur est le ratio signal/bruit. Dans java c'est facile à implémenter ce calcul par :

```
1 double psnr = 10.0 * Math.log(maxSignal*maxSignal/mse) / Math.log(10.0);
```

Il ne prend pas en compte la qualité visuelle de reconstruction parce qu'il est utilisé pour mesurer la proximité entre l'image compressée et l'image originale au niveau du signal.

Pour conclure, on peut évaluer la qualité d'une reconstruction d'image en calculant son PSNR. Plus le PSNR est grand plus l'image reconstruite est de bonne qualité.

3 Sous-échantillonnage d'une image

Le ré-échantillonnage est une modification des dimensions ou de la résolution d'une image qui provoquera un changement dans le nombre de pixels qu'elle contient.

Lorsqu'on parle de sous-échantillonnage, on parle d'une image qui contient moins de pixels que l'image de départ. La diminution du nombre de pixels donne une perte de la qualité du détail[5].

C'est aussi important de savoir qu'on utilise le ré-échantillonnage toujours après avoir fixé la résolution de l'image et ses dimensions à l'impression. Comme dans ce projet, on s'intéresse aussi au calcul de PSNR qui a besoin d'avoir deux images de la même taille (dimension en pixels). Dans ce cas là, ça sera les dimensions à l'impression (taille réelle) et sa DPI qui vont être changés mais pas les tailles en pixels.

Donc dans ce projet, un sous-échantillonnage peut être utile pour réduire la taille et le poids d'une image dont la qualité est suffisamment élevée pour le supporter[2].

L'affichage d'une image sur un écran ou du papier dépend de la résolution du support utilisé. La résolution est la densité de pixels affichés. Elle est donnée en dots per inch (ou pixels par pouce) :

- Écrans : 72 DPI (ou PPP)
- Imprimantes : 300 DPI (ou PPP)

Donc ici dans ce projet, je propose deux moyens pour sous-échantillonner une image. Avec la première façon, on arrive à réduire la taille réelle de l'image de départ en modifiant sa DPI sans changer sa taille en pixel. J'ai implémenté aussi une deuxième façon en réduisant ou agrandissant sa taille en pixel avec une méthode d'interpolation.

3.1 Moyen 1 : Réduction d'une image en modifiant sa DPI

La résolution d'une image représente une relation entre le nombre de pixels qui composent l'image et les dimensions de cette image.

Donc une image possède 3 caractéristiques(cf : Figure 7) :

* Sa taille en pixels
* Ses dimensions réelles (taille réelle)
* Sa résolution (DPI).

$$Résolution\ en\ pixels\ par\ pouce = \frac{nombre\ de\ pixels}{(taille\ réelle\ en\ cm / 2{,}54)}$$

Figure 7 – *DPI*

Pour moi,faire un échantillonnage de l'image est une opération qui consiste à augmenter ou réduire le nombre de pixel constituant l'image. Quand les pixels sont supprimés, on perd les informations, ce qui s'appellent aussi un sous-échantillonnage. Ça répond aussi au but du projet.

Dans ce projet, on va calculer le PSNR de l'image modifiée pour le mettre en correspondance avec l'image de départ. Pour effectuer un calcul PSNR, il faut avoir deux images avec la même taille en pixel. Comment peut-on réduire une image sans modifier sa taille en pixel ? La réponse est qu'on peut augmenter sa DPI lorsque les pixels sont imprimés plus petits, l'image est réduite par rapport sa taille réelle.

3.2 Moyen 2 : Réduction d'une image en modifiant sa taille en pixel avec l'interpolation

En utilisant cette moyen, on arrive à dégrader une image en réduisant ou agrandissant sa taille en pixel et aussi appliquer une méthode d'interpolation.

Interpolation bilinéaire

L'interpolation bilinéaire est une méthode d'interpolation pour les fonctions de 2 variables sur une grille régulière. Elle permet de calculer la valeur d'une fonction en un point quelconque, à partir de ses 2 plus proches voisins dans chaque direction [WWW7].

Dans le cas de traitement d'images, les pixels de l'image modifiée seront calculés par interpolation bilinéaire avec les 4 pixels voisins(cf : Figure 8).

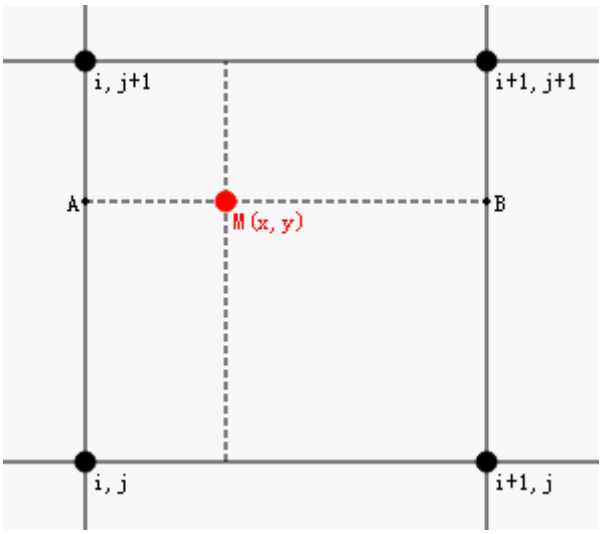

Figure 8 – *Interpolation bilinéaire*

Interpolation bicubique

L'interpolation bicubique a pour but d'interpoler un ensemble de points distribués sur une grille régulière. La surface interpolée est plus lisse que les surfaces correspondantes obtenues par interpolation bilinéaire.

Dans le domaine du traitement d'images numériques, l'interpolation bicubique est souvent préférée à une interpolation bilinéaire ou à la technique du plus proche voisin pour le ré-échantillonnage d'images [WWW4].

3 Détail d'implémentation et résultats d'expérimentation

1 Librairie ImageJ et les méthodes utilisées

Avant de commencer à programmer dans ImageJ, il vaut mieux comprendre ce qu'est une programmation orientée objet. Comme j'ai réalisé un plugin d'ImageJ dans ce projet, c'est important de savoir que les plugins sont des classes Java qui sont placées dans un certain dossier, le dossier "plugins" d'ImageJ ou un de ses sous-dossiers[WWW3][1].

Pour créer un plugin, la classe principale java doit contenir un "_" dans son nom et il y a deux type de PlugIn dans ImageJ :

* La classe PlugIn : pas besoin d'image en entrée
* La classe PlugInFilter : besoin d'image en entrée

ImageJ API

Dans les programmes d'ImageJ, la classe principale est ij. Ici je vous présente quelques classes importantes qui sont utilisées dans mon programme :

Classe ij :

- ImageJ : La classe principale d'une application ImageJ qui contient la méthode run, point d'entrée du programme.
- ImagePlus : La représentation d'une image dans ImageJ qui base sur un ImageProcessor.
- ImageStack : Une pile d'images de taille variable.
- WindowManager : L'ensemble des fenêtres ouvertes.

Classe ij.gui :

- GenericDialog : Une boîte de dialogue modulaire pour communiquer avec l'utilisateur.
- ProgressBar : Une barre qui contient les informations de la progression d'une opération faite par utilisateur.
- ImageWindow : Une fenêtre pour afficher une image, dérivée de java.awt.Frame.
- PlotWindow : Un ImageWindow pour les tracés de courbes.

Classe ij.io : Un paquet de gestion des entrées sorties qui permet lire et écrire les images.

Classe ij.process :

- ImageConverter : Une classe contenant les méthodes pour convertir les images d'un type dans un autre.
- ImageProcessor : Un imageProcessor contient les méthodes permettant de travailler réellement sur les données contenues dans l'image.
- ImageWindow : Une fenêtre pour afficher une image, dérivée de java.awt.Frame.
- PlotWindow : Un ImageWindow pour les tracés de courbes.

Créer ou ouvrir une image

Il y a beaucoup de façons pour créer une image dans ImageJ, on peut aussi récupérer l'image actuelle qui est couverte dans la fenêtre.

```
ImagePlus imp = WindowManager.getCurrentImage();
ou
ImagePlus createImagePlus();
ou
ImageProcessor createProcessor(int width, int height);
```

- IJ.getImage() : Renvoie une référence à l'image active, ou affiche un message d'erreur et annule le plugin ou macro si aucune image sont ouvertes
- ij.ImagePlus.getTitle() : Renvoie le nom de l'image

Récupérer les pixels

Pour travailler sur les valeurs de l'image, il faut récupérer la valeur de chaque pixel, on peut utiliser pour cela la méthode java.lang.Object getPixels() de l' ImageProcessor.

```
    int nx = impRef.getWidth();
    int ny = impRef.getHeight();

    s =  ipRef.getPixelValue(x, y);
    t = ipTest.getPixelValue(x, y);
```

Après avoir récupéré les valeurs des pixels, on peut travailler sur les analyses ou les calculs comme le PSNR dans ce projet.

Modifier la facteur de qualité

J'ai essayé de modifier le facteur de qualité de l'image départ avec deux moyens différents. Soit on peut suivre les étapes de la compression JPEG qui était présenté dans la chapitre 1. Mais ça n'a peut être pas d'avantage pour traiter un gros volume de données. Soit la classe imageJ ij.io.FileSaver propose deux fonctions qui permettent de modifier la qualité d'une image facilement.

- FileSaver.saveAsJpeg(String path) : Enregistrer l'image au format JPEG en utilisant le chemin d'accès spécifié.
- FileSaver.setJpegQuality(int quality) : Indique la qualité d'image (0-100). 0 est la plus pauvre qualité d'image, la plus haute compression, et 100 est la meilleure qualité d'image, la compression la plus bassse.
- FileSaver.getJpegQuality() : Renvoie le paramètre de qualité JPEG actuelle (0-100).
- ImageWriteParam.setCompressionQuality(float quality)
- ImageWriteParam.setCompressionMode(int mode)

Sinon on peut aussi utiliser une classe java ImageWriteParam et méthode getDefaultWriteParam() pour renvoyer un nouvel objet ImageWriteParam du type approprié pour ce format de fichier contenant des valeurs par défaut,

```
ImageWriteParam param = writer.getDefaultWriteParam();
param.setCompressionMode(param.MODE_EXPLICIT);
param.setCompressionQuality(quality/100f);
```

Dessiner une courbe

ImageJ permet de dessiner une courbe d'analyse facilement. Dans la classe ij.gui.Plot il y a une méthode Plot(String title, String xLabel, String yLabel, float[] xValues, float[] yValues) qui permet de construire une nouvelle PlotWindow, les paramètres (color, lineWidth) sont pris en compte au moment où «Draw» ou «getProcessor» est appelé. Les données sont représentées graphiquement en tant que ligne.

```
Plot plot = new Plot("Courbe","X-FQ","Y-PSNR",x,y);
  plot.setLimits(1, 200, 1, 100);
  plot.setLineWidth(2);
  plot.addPoints(x, y, 7);
  plot.setColor(Color.blue);
  plot.show();
```

Modifier le DPI d'une image

Le JPEG File Interchange Format, abrégé JFIF, est une norme pour faciliter l'échange de fichiers contenant des images enregistrées avec la compression JPEG.

Il faut noter que les informations JFIF au sein d'une image se présentent sous la forme d'un bloc de données, un segment, situé vers le début du fichier JPEG.

Segement	Taille
Code (marker) APP0	2
Longueur	2
Identifiant	5
Version	2
Unité de densité	1
Densité X (horizontal)	2
Densité Y (vertical)	2
tw (thumbnail width)	1
th (thumbnail height)	1
Thumbnail data	3 × *tw* × *th*

Figure 1 – *JFIF*

Sur le tableau (cf : Figure 1), on peut trouver que la position 13 et 14 contiennent la valeur d'DPI d'une image. C'est à dire qu'on va modifier la treizième et quatorzième valeur dans le segment.

```
    RandomAccessFile raf = new RandomAccessFile(file, "rw");
  //read the begining 11 bytes
  byte [] tag = new byte[11];
  //the reference heading
  byte [] refTag = {
  (byte)0xFF,(byte)0xD8,(byte)0xFF,(byte)0xE0,
  (byte)0x00,(byte)0x10,(byte)0x4A,(byte)0x46,
  (byte)0x49,(byte)0x46,(byte)0x00};
  //read the begining 11 bytes
  int l = raf.read(tag);
  //test if this is a JFIF tag
  if(Arrays.equals(tag,refTag)){
  raf.seek(13);
  raf.writeByte(1);
  raf.writeShort(dpi);
  raf.writeShort(dpi);
```

Après avoir modifié la DPI d'une image de départ et sans toucher sa taille en pixel, on peut sauvegarder cette image modifiée en utilisant la classe ij.io.FileSaver.saveAsJpeg(String path) avec les paramètres par

défauts, par exemple sa qualité et mode de compression. Donc dans ce cas là, ça sera la taille réelle en centimètre de cette image qui vont être changé. Ensuite on peut effectuer le calcul PSNR avec cette image modifiée et l'image de départ qui ont la même taille en pixel.

Dégrader un image avec l'interpolation

C'est aussi un deuxième moyen pour sous-échantillonner une image. Dans la classe ij.process.ImageProcessor il y a une méthode setInterpolationMethod(int method) qui permet de définir la méthode d'interpolation (NONE, BILINEAR ou BICUBIC) utilisée par différente fonctionnalité ImageJ comme échelle (), redimensionner () et tourner ().

J'ai utilisé une liste déroulante qui permet de choisir la méthode d'interpolation parmi :

- NONE
- BILINEAR
- BICUBIC

Une méthode resize(int dstWidth, int dstHeight) permet de sous-échantillonner ou sur-échantillonner l'image de départ en modifiant sa taille en pixel.

```
1 image_ip.setInterpolationMethod(interpolation);
2 ImageProcessor roiResize_ip=image_ip.resize(value_w, value_h);
```

2 Résultats d'expérimentation

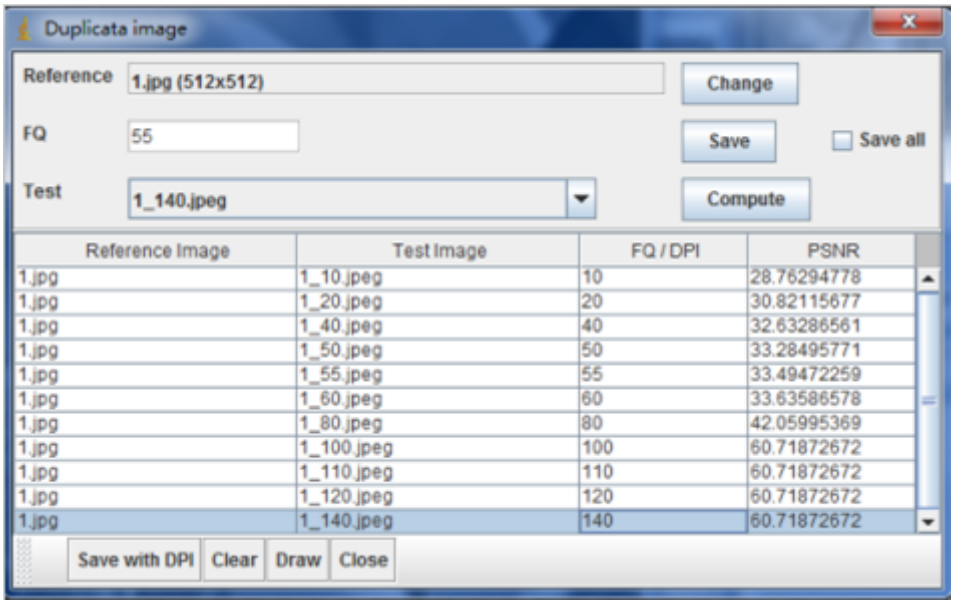

Figure 2 – *Fenêtre principale avec résultats PSNR*

Pour tester mon plugin, j'ai utilisé une image lena.jpg avec la taille en pixel 512x512 de type 8 bits, parce que cette image rend les résultats (changement d'image) plus visible et plus évident.

Pour lancer le plugin, il faut ouvrir une ou plusieurs image de référence. Et puis on va dans le menu ImageJ>Plugins>Compile and run>Mon plugin.

Réduction de la facteur qualité et le calcul PSNR

Toutes les image ouvertes vont apparaître dans la liste 'Reference image' et la liste 'Test image'. On doit choisir une image dans cette liste comme l'image de départ et l'image test (cf : Figure 3).

A droite de la fenêtre principale, il y a 3 boutons (cf : Figure 2) :

- Change : changer l'image de départ
- Save : modifier et sauvegarder l'image de départ avec le facteur qualité saisi dans le champs 'FQ' sous le 'NomImageDépart_FQ.jpg'

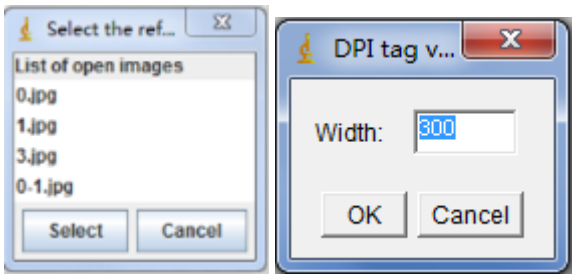

Figure 3 – *Liste de l'image et Changement DPI*

- Compute : calculer le PSNR avec l'image de départ choisie et l'image test (image modifiée)

Et la case à cocher 'Save all' permet de traiter plusieurs images en même temps et les sauvegarder dans un dossier.

Après avoir effectué un calcul PSNR, les résultats seront imprimées sur l'écran avec le facteur de qualité de l'image test (cf : Figure 2). Les images modifiées seront sauvegardées en dur dans un répertoire codé dans le programme (cf : Figure 4). Avec une comparaison des deux images, on peut dire que la qualité de l'image modifiée est réduite avec le facteur de qualité = 10.

Figure 4 – *FQ = 90 et FQ = 10*

Avec les résultats obtenues en cliquant le bouton 'DRAW' en bas de la fenêtre principale, on peut faire une courbe avec deux dimensions : facteur de qualité et PSNR (cf : Figure 5).

Analyse de résultat : Comme le facteur de qualité est compris entre 1 et 100, on observe sur le graphique de courbe ses valeurs augmenter, ainsi que celles du PSNR. Jusqu'à le facteur de qualité = 100, le PSNR s'arrête augmenter. Parce que la valeur maximale du facteur de qualité est 100, la qualité d'une image ne changera plus si le facteur de qualité est dépassé 100.

si le PSNR est fixé à une valeur, il fournit une compression compression minimale et une qualité maximale). Si le PSNR = 'Infinite', c'est à dire qu'on est en train de comparer deux même images avec la même taille en pixel, parce que le EQM = 0. Ça nous rend aussi une méthode de détecter les duplicatas d'une image référence.

Dégradation avec le DPI

En bas de la fenêtre principale, on trouve aussi 4 boutons :

- Save with DPI : lancer une fenêtre qui permet de saisir la valeur d'DPI souhaitée et sauvegarder
- Clear : effacer tous les résultats affichés
- Draw : lancer et faire une courbe de résultats
- Close : sortie de l'application

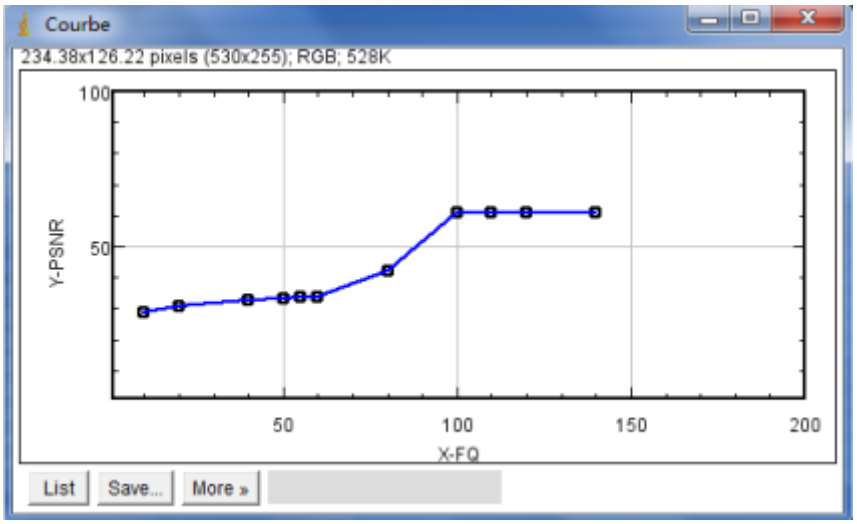

Figure 5 – *Courbe de la facteur qualité et PSNR*

En cliquant sur le bouton 'Save with DPI', une nouvelle fenêtre apparaît qui permet de modifier le DPI de l'image de départ sans toucher sa taille en pixel (cf : Figure 3). Pour comparer avec l'image modifiée par DPI, on peut dire que l'image est dégradée au niveau de sa taille réelle (cf : Figure 6).

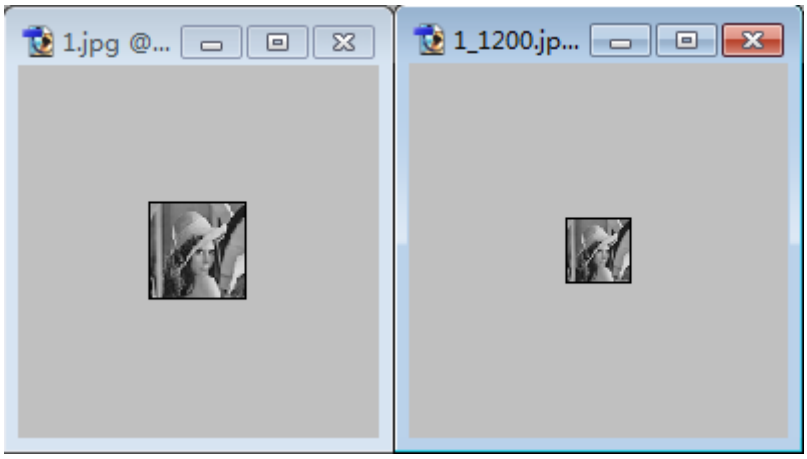

Figure 6 – *Dégradation sur la taille réelle*

Dégradation avec l'interpolation

Après avoir lancé ce plugin dans ImageJ, on arrive sur la fenêtre principale (cf : Figure 6) qui permet de modifier la taille en pixel de l'image de départ et la sauvegarder avec une méthode d'interpolation choisie (cf : Figure 7).

Mais je n'ai pas trouvé une façon de calculer le PSNR après avoir dégradé l'image en modifiant sa taille en pixel. Parce que le calcul PSNR travaille sur les pixels, c'est à dire qu'il nous demande d'avoir deux images avec la même taille en pixel. Donc j'ai pensé à dégrader une image départ en utilisant les 3 méthode d'interpolation et la même taille en pixel. Comme ça on arrivera à calculer le PSNR de l'image dégradée avec interpolation Bicubic et de l'image dégradée avec interpolation Bilinaire. c'est à dire que (cf : Figure 8) :

- Image à gauche 256x256 : image dégradée par 50% avec aucune méthode d'interpolation
- Image au centre 256x256 : image dégradée par 50% avec méthode d'interpolation Bilinaire
- Image à droite 256x256 : image dégradée par 50% avec méthode d'interpolation Bicubic

Visuellement on ne voit pas trop la différence de qualité entre l'image de départ et l'image dégradée excepté leurs tailles en pixel. Mais l'algorithme bicubique préserve les détails fins mieux que l'algorithme bilinéaire au niveau des contours. Et puis j'ai calculé le PSNR sur les deux images avec l'image référence qui porte aucune méthode d'interpolation. Voici la résultats de la valeur PSNR sur Bilinaire et Bicubic :

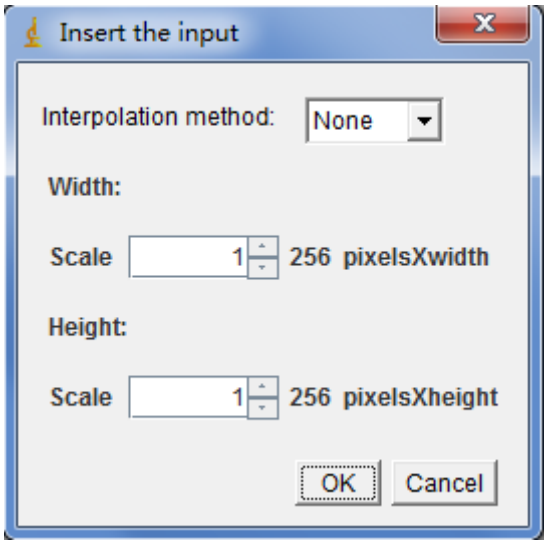

Figure 7 – *Fenêtre principale de dégradation*

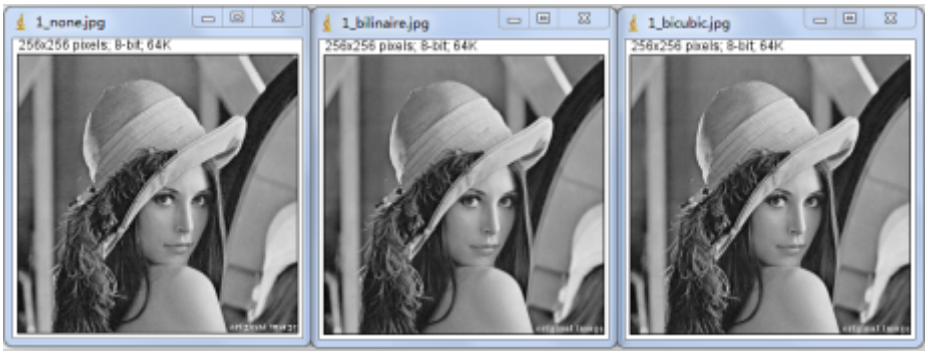

Figure 8 – *Comparaision de méthode d'interpolation : None, Bilinaire, Bicubic*

- PSNR de Bilinaire = 28.0948
- PSNR de Bicubic = 27.8989

On peut dire que même si visuellement on ne voit pas trop la différence entre les deux méthode d'interpolation. Mais le résultat de PSNR s'exprime que le PSNR de Bilinaire est un peu plus élevé que le PSNR de Bicubic.

C'est à dire que la dégradation avec Bilinaire nous rend une image avec une mieux qualité que la dégradation avec Bicubic, parce qu'on peut évaluer la qualité d'une dégradation d'image en calculant son PSNR. Plus le PSNR est grand plus l'image dégradée est de bonne qualité.

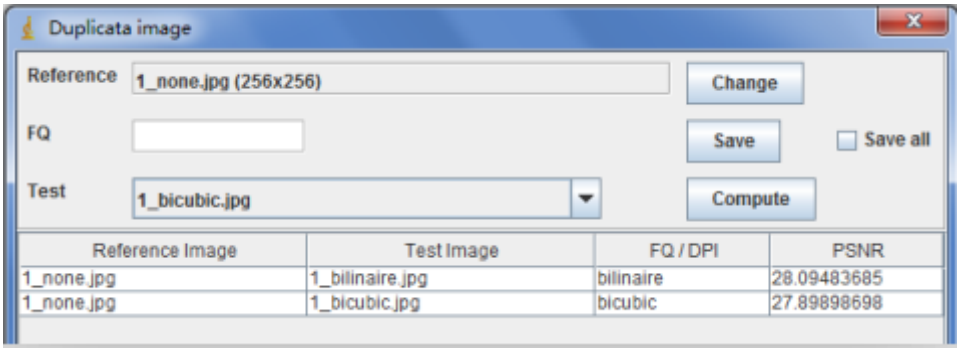

Figure 9 – *Résultat PSNR : Bilinaire, Bicubic*

4 Conclusion

Ce projet m'a répondu beaucoup de question dans le domaine de traitement d'image et appris certains mécanismes au niveau de la compression et la dégradation d'une image :

- Processus de la compression JPEG
- Calcul PSNR en mesurant la qualité d'une image
- Re-échantillonnage d'une image
- Méthode d'interpolation : Bilinaire et Bicubic
- Utilisation de la librairie ImageJ dans l'environnement Java

En plus, je trouve qu'il y a beaucoup d'ambiguïté de définition dans ce domaine. Il faut vraiment travailler fort si on veut bien comprendre les définitions et les mécanismes. En faisant ce projet, au début j'ai eu un peu de mal à comprendre quelques parties du sujet par rapport les nouveaux termes, les définitions et les ambiguïtés dans le domaine de traitement d'image. Mais après avoir compris ce qu'il faut faire, la partie réalisation en utilisant librairie ImageJ et java n'était pas difficile.

Annexes

PSNR.java permet de recompresser une image en modifiant le facteur de qualité et dégrader une image en changeant sa DPI.

```
package manga109;
import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;
import ij.gui.GUI;
import ij.gui.GenericDialog;
import ij.gui.Plot;
import ij.gui.PlotWindow;
import ij.io.FileSaver;
import ij.io.SaveDialog;
import ij.plugin.BrowserLauncher;
import ij.plugin.PlugIn;
import ij.process.ImageProcessor;
...

public class PSNR_ implements PlugIn, ActionListener, MouseMotionListener {

  /** List of test images open in ImageJ */
  private Vector<TestImage> testList;

  /** Image of reference */
  private TestImage imageref;

  private GridBagLayout     layout      = new GridBagLayout();
  private GridBagConstraints  constraint    = new GridBagConstraints();
  private JButton       bnChange    = new JButton("Change");
  private JButton       bnCompute   = new JButton("Compute");
  private JButton       bnSave        = new JButton("Save");
  private JButton       bnSave2       = new JButton("Save with DPI");

  private JButton       bnClose     = new JButton("Close");
  private JButton       bnDraw      = new JButton("Draw");
  private JButton       bnClear     = new JButton("Clear");
  private JCheckBox       bnCheck     = new JCheckBox("Save all");

  private JLabel        lblRef      = new JLabel("01234567890123456789012345678901234567890123456789");
  private JTextArea           txfSav          = new JTextArea(1, 10);
  private JComboBox       cmbTest     = new JComboBox();
  private DefaultTableModel model     = new DefaultTableModel();
  private JDialog       dialog;
  private JPanel        pnImages;

  /**
   * Implements the run method of PlugIn.
   * Starting point of the plugin.
   */
  public void run(String arg) {
    if (IJ.versionLessThan("1.21a")) {
      return;
    }
    ImagePlus imp = WindowManager.getCurrentImage();
    if (imp == null) {
      IJ.error("No 8-bit, 16-bit, or 32-bit images are open.");
      return;
    }
    imageref = new TestImage(imp);
    if (!imageref.isType()) {
      IJ.error("No 8-bit, 16-bit, or 32-bit images are open.");
      return;
    }
```

```
    doDialog(imageref);
  }

  String saveAsJpeg(ImagePlus imp, String path, int quality) {
    int width = imp.getWidth();
    int height = imp.getHeight();
    int biType = BufferedImage.TYPE_INT_RGB;
    boolean overlay = imp.getOverlay()!=null && !imp.getHideOverlay();
    if (imp.getProcessor().isDefaultLut() && !imp.isComposite() && !overlay)
      biType = BufferedImage.TYPE_BYTE_GRAY;
    BufferedImage bi = new BufferedImage(width, height, biType);
    String error = null;
    try {
      Graphics g = bi.createGraphics();
      Image img = imp.getImage();
      if (overlay)
        img = imp.flatten().getImage();
      g.drawImage(img, 0, 0, null);
      g.dispose();
      Iterator iter = ImageIO.getImageWritersByFormatName("jpeg");
      ImageWriter writer = (ImageWriter)iter.next();
      File f = new File(path);
      String originalPath = null;
      boolean replacing = f.exists();
      if (replacing) {
        originalPath = path;
        path += imp.getShortTitle()+ "_" + txfSav.getText() + ".jpeg";
        f = new File(path);
      }
      ImageOutputStream ios = ImageIO.createImageOutputStream(f);
      writer.setOutput(ios);
      ImageWriteParam param = writer.getDefaultWriteParam();
      param.setCompressionMode(param.MODE_EXPLICIT);
      param.setCompressionQuality(quality/100f);
      if (quality == 100)
        param.setSourceSubsampling(1, 1, 0, 0);
      IIOImage iioImage = new IIOImage(bi, null, null);
      writer.write(null, iioImage, param);
      ios.close();
      writer.dispose();
      if (replacing) {
        File f2 = new File(originalPath);
        boolean ok = f2.delete();
        if (ok) f.renameTo(f2);
      }
    } catch (Exception e) {
      error = ""+e;
      IJ.error("Jpeg Writer", ""+error);
    }
    return error;
  }

  void saveAsJpegWithDpi(ImagePlus imp) {
    //open DPI dialog
    int dpi = 300;
    GenericDialog dialog = new GenericDialog("DPI tag value:");
    dialog.addNumericField("Width: ", dpi, 0);
    dialog.showDialog();
    dpi=(int)dialog.getNextNumber();

    String dir = "", name = "";
```

```
122     if (imp!=null) {
123     SaveDialog sd = new SaveDialog("Save as jpg", imp.getTitle(), ".jpg");
124     dir = sd.getDirectory();
125     name = sd.getFileName();
126     } else {
127
128     File tmpfile = new File("D:\\test\\");
129     if (tmpfile.isDirectory()) {
130     dir = "D:\\test\\";
131     name = imp.getTitle();
132     } else {
133     dir = tmpfile.getParent();
134     name = tmpfile.getName();
135     }
136     }
137     if (name == null || name == "") {
138     return;
139     }
140
141     //Save as JPG
142     File file = new File(dir,name);
143     FileSaver fs = new FileSaver(imp);
144     if(!fs.saveAsJpeg(file.getPath())){
145     IJ.showMessage("fail to save file!");
146     return;
147     }
148
149     try {
150     RandomAccessFile JFIF = new RandomAccessFile(file, "rw");
151     //read the begining 11 bytes
152     byte [] tag = new byte[11];
153     //the reference heading
154     byte [] refTag = {
155     (byte)0xFF,(byte)0xD8,(byte)0xFF,(byte)0xE0,
156     (byte)0x00,(byte)0x10,(byte)0x4A,(byte)0x46,
157     (byte)0x49,(byte)0x46,(byte)0x00};
158     //read the begining 11 bytes
159     int l = JFIF.read(tag);
160     //test if this is a JFIF tag
161     if(Arrays.equals(tag,refTag)){
162     JFIF.seek(13);
163     JFIF.writeByte(1);
164     JFIF.writeShort(dpi);
165     JFIF.writeShort(dpi);
166     }else{
167     IJ.showMessage("Can not find JFIF tag, fail to write DPI info!");
168     }
169
170     JFIF.close();
171
172     } catch (IOException e) {
173     System.out.println("IOException:");
174     e.printStackTrace();
175     }
176
177   }
178
179   /**
180   * Build the dialog box.
181   */
182   private void doDialog(TestImage imageref) {
```

```
183
184     dialog = new JDialog(IJ.getInstance(), "Duplicata image");
185     dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
186
187     JToolBar tool = new JToolBar("Information on PSNR");
188     tool.addSeparator();
189     tool.addSeparator();
190     tool.add(bnSave2);
191     tool.add(bnClear);
192     tool.add(bnDraw);
193     tool.add(bnClose);
194
195     cmbTest.addItem("01234567890123456789012345678901234567890123456789");
196     lblRef.setBorder(BorderFactory.createEtchedBorder());
197     lblRef.setText("01234567890123456789012345678901234567890123456789");
198
199     txfSav.setBorder(BorderFactory.createEtchedBorder());
200
201     model.addColumn("Reference Image");
202     model.addColumn("Test Image");
203     model.addColumn("FQ / DPI");
204     model.addColumn("PSNR");
205
206     JTable table = new JTable(model);
207     table.setShowGrid(true);
208     PSNRPane scrollPane = new PSNRPane(table);
209
210     scrollPane.setPreferredSize(new Dimension(220, 200));
211     TableColumn col1 = table.getColumnModel().getColumn(0);
212     col1.setPreferredWidth(150);
213     TableColumn col2 = table.getColumnModel().getColumn(1);
214     col2.setPreferredWidth(150);
215
216     pnImages = new JPanel(layout);
217     pnImages.setBorder(BorderFactory.createEtchedBorder());
218     addComponent(pnImages, 0, 0, 1, 1, 5, new JLabel("Reference"));
219     addComponent(pnImages, 0, 1, 1, 1, 5, lblRef);
220     addComponent(pnImages, 0, 2, 1, 1, 5, bnChange);
221
222     addComponent(pnImages, 1, 0, 1, 1, 5, new JLabel("FQ"));
223     addComponent(pnImages, 1, 1, 1, 1, 5, txfSav);
224     addComponent(pnImages, 1, 2, 1, 1, 5, bnSave);
225     addComponent(pnImages, 1, 3, 1, 1, 5, bnCheck);
226
227     addComponent(pnImages, 2, 0, 1, 1, 5, new JLabel("Test"));
228     addComponent(pnImages, 2, 1, 1, 1, 5, cmbTest);
229     addComponent(pnImages, 2, 2, 1, 1, 5, bnCompute);
230
231     bnCheck.addActionListener(this);
232     bnCompute.addActionListener(this);
233     bnChange.addActionListener(this);
234     bnSave.addActionListener(this);
235     pnImages.addMouseMotionListener(this);
236
237     bnClose.addActionListener(this);
238     bnDraw.addActionListener(this);
239     bnClear.addActionListener(this);
240     bnSave2.addActionListener(this);
241
242     JPanel panel1 = new JPanel(new BorderLayout());
243     panel1.add(pnImages, BorderLayout.NORTH);
```

```
244     panel1.add(scrollPane, BorderLayout.CENTER);
245     panel1.add(tool, BorderLayout.SOUTH);
246 
247     dialog.add(panel1);
248     dialog.setResizable(true);
249     dialog.pack();
250 
251     dialog.setMinimumSize(dialog.getSize());
252 
253     lblRef.setMinimumSize(lblRef.getSize());
254     cmbTest.setMinimumSize(cmbTest.getSize());
255     lblRef.setPreferredSize(lblRef.getSize());
256     cmbTest.setPreferredSize(cmbTest.getSize());
257 
258     lblRef.setBounds(lblRef.getBounds());
259     cmbTest.setBounds(cmbTest.getBounds());
260 
261     dialog.pack();
262     dialog.setVisible(true);
263 
264     lblRef.setText(imageref.getInfoName());
265     testList = getImageList(imageref);
266     makeImageList(testList, cmbTest);
267 
268   }
269 
270   /**
271   * Add a component in a panel in the northeast of the cell.
272   */
273   final private void addComponent(JPanel pn, int row, int col, int width, int height, int ↩
          space, JComponent comp) {
274     constraint.gridx = col;
275     constraint.gridy = row;
276     constraint.gridwidth = width;
277     constraint.gridheight = height;
278     constraint.anchor = GridBagConstraints.NORTHWEST;
279     constraint.insets = new Insets(space, space, space, space);
280     layout.setConstraints(comp, constraint);
281     pn.add(comp);
282   }
283 
284   /**
285   * Implements the actionPerformed for the ActionListener.
286   */
287   public synchronized void actionPerformed(ActionEvent e) {
288 
289     if (e.getSource() == bnClose) {
290       dialog.dispose();
291     }
292     else if (e.getSource() == bnClear){
293 
294       for( int i = model.getRowCount() - 1; i >= 0; i-- ) {
295             model.removeRow(i);
296       }
297     }
298 
299       else if (e.getSource() == bnDraw){
300 
301         int row = model.getRowCount();
302         int col = model.getColumnCount();
303 
```

```
304           float[] x = new float[row];
305           float[] y = new float[row];
306 
307           for(int i = 0; i < row; i++) {
308             x[i] = Float.parseFloat(model.getValueAt(i, 2).toString());
309           }
310 
311           for(int i = 0; i < row; i++) {
312             y[i] = Float.parseFloat(model.getValueAt(i, 3).toString());
313           }
314 
315             PlotWindow.noGridLines = false; // draw grid lines
316 
317       Plot plot = new Plot("Courbe","X-FQ","Y-PSNR",x,y);
318             plot.setLimits(1, 200, 1, 100);
319             plot.setLineWidth(2);
320             plot.addPoints(x, y, 7);
321 
322             plot.setColor(Color.blue);
323             plot.show();
324     }
325     else if (e.getSource() == bnCompute) {
326       PSNR();
327     }
328     else if (e.getSource() == bnChange) {
329       Select selectDialog = new Select();
330       String name = selectDialog.getSelected();
331       ImagePlus imp = WindowManager.getImage(name);
332       if (imp != null) {
333         imageref = new TestImage(imp);
334         lblRef.setText(imageref.getInfoName());
335         testList = getImageList(imageref);
336         makeImageList(testList, cmbTest);
337       }
338     }
339     else if (e.getSource() == bnSave){
340 
341             if (bnCheck.isSelected()){
342               String dir1 = "D:\\test\\all\\";
343               String dir2 = "D:\\test\\all\\";
344 
345             String[] list = new File(dir1).list();
346             if(list == null) return;
347             for(int i = 0; i < list.length; i++){
348               boolean isDir = (new File(dir1 + list[i])).isDirectory();
349               if (!isDir && !list[i].startsWith(".")) {
350                   int fq = Integer.parseInt(txfSav.getText());
351                   if (fq > 100) fq = 100;
352                   if (fq < 0) fq = 0;
353 
354                 ImagePlus img = IJ.openImage(dir1+list[i]);
355                 WindowManager.setTempCurrentImage(img);
356                 saveAsJpeg(img, dir2, fq);
357               }
358                }
359           }
360 
361       int fq = Integer.parseInt(txfSav.getText());
362       if (fq > 100) fq = 100;
363       if (fq < 0) fq = 0;
364       String pathtemp = "D:\\test\\";
```

```
365       ImagePlus impRef1 = imageref.getImagePlus();
366       saveAsJpeg(impRef1, pathtemp, fq);
367
368     }
369
370         else if (e.getSource() == bnSave2){
371
372           ImagePlus impRef1 = imageref.getImagePlus();
373           saveAsJpegWithDpi(impRef1);
374
375     }
376
377     notify();
378   }
379   public synchronized void itemStateChanged(ItemEvent e) {
380     PSNR();
381   }
382
383   /**
384   * Implements the methods for the MouseMotionListener.
385    */
386   public synchronized void mouseDragged(MouseEvent e) {}
387   public synchronized void mouseMoved(MouseEvent e) {
388     if (e.getSource() == pnImages) {
389       testList = getImageList(imageref);
390       makeImageList(testList, cmbTest);
391     }
392   }
393
394   /**
395   * Set the list of images into the ComboBox.
396   */
397   public void makeImageList(Vector<TestImage> list, JComboBox cmb) {
398
399     bnCompute.setEnabled(false);
400     String selected = (String)cmb.getSelectedItem();
401
402     if (imageref.getImagePlus() == null) {
403       lblRef.setText("Reference image is invalid.");
404       return;
405     }
406     if (imageref.getImagePlus().getProcessor() == null) {
407       lblRef.setText("Reference image is invalid.");
408       return;
409     }
410
411     int n = cmb.getItemCount();
412     for (int i=0; i<n; i++)
413       cmb.removeItemAt(0);
414
415     if (list.size() <= 0) {
416       cmb.addItem("No 8-bit, 16-bit, or 32-bit images are open.");
417       cmb.setEnabled(false);
418       return;
419     }
420     else {
421       for (int i=0; i<list.size(); i++)
422         cmb.addItem(list.get(i).getShortName());
423       cmb.setEnabled(true);
424       bnCompute.setEnabled(true);
425     }
```

```
426     if (selected == null) {
427       cmb.setSelectedIndex(0);
428     }
429     else {
430       cmb.setSelectedItem(selected);
431     }
432   }
433
434   /**
435    * Scan the grayscale image open in ImageJ and build a list of images.
436    */
437   public Vector<TestImage> getImageList(TestImage imageref) {
438     Vector<TestImage> list = new Vector<TestImage>();
439     int[] ids = WindowManager.getIDList();
440     if (imageref.getImagePlus() != null)
441       if (ids != null) {
442       for (int i=0; i<ids.length; i++) {
443         ImagePlus imp = WindowManager.getImage(ids[i]);
444         if (imp != null) {
445           TestImage ti = new TestImage(imp);
446           if (ti.isType())
447           if (ti.isSameSize(imageref))
448             list.add(ti);
449         }
450       }
451     }
452     return list;
453   }
454
455   /**
456   * Call the PSNR computation.
457   */
458   private void PSNR() {
459
460     ImagePlus impRef = imageref.getImagePlus();
461     if (impRef == null) {
462       IJ.error("The reference image is not valid.");
463       return;
464     }
465     if (impRef.getProcessor() == null) {
466       IJ.error("The reference image is not valid.");
467       return;
468     }
469
470     int indexTest = cmbTest.getSelectedIndex();
471     ImagePlus testImp = testList.get(indexTest).getImagePlus();
472     if (testImp == null) {
473       IJ.error("The test image is not valid.");
474       return;
475     }
476
477     if (testImp.getProcessor() == null) {
478       IJ.error("The test image is not valid.");
479       return;
480     }
481
482     if (impRef.getWidth() != testImp.getWidth()) {
483       IJ.error("Not the same width ("+impRef.getTitle()+","+testImp.getTitle()+")");
484       return;
485     }
486
```

```
487     if (impRef.getHeight() != testImp.getHeight()) {
488       IJ.error("Not the same height ("+impRef.getTitle()+","+testImp.getTitle()+")");
489       return;
490     }
491
492     int nzr = impRef.getStack().getSize();
493     int nzt = testImp.getStack().getSize();
494     if (nzr != 1 && nzt != 1) {
495       if (nzr != nzt) {
496         IJ.error("Not the same number of slices ↩
                 ("+impRef.getTitle()+","+testImp.getTitle()+")");
497         return;
498       }
499     }
500
501     int nx = impRef.getWidth();
502     int ny = impRef.getHeight();
503     double chrono = System.currentTimeMillis();
504     for (int z=1; z<=Math.max(nzr, nzt); z++) {
505       String row[] = new String[4];
506       IJ.showStatus("Compute PSNR " + z + " " + Math.max(nzr, nzt));
507       int ir = Math.min(z, nzr);
508       int it = Math.min(z, nzt);
509       ImageProcessor ipTest = testImp.getStack().getProcessor(it);
510       ImageProcessor ipRef  = impRef.getStack().getProcessor(ir);
511       row[0] = impRef.getTitle();
512       row[1] = testImp.getTitle();
513       double maxSignal = -Double.MAX_VALUE;
514       double s, t, mse = 0.0, mae = 0.0, es = 0.0, ms = 0.0;
515       int N = 0;
516       for (int y=0; y<nx; y++)
517       for (int x=0; x<ny; x++) {
518         s =  ipRef.getPixelValue(x, y);
519         if (s > maxSignal)
520           maxSignal = s;
521         t = ipTest.getPixelValue(x, y);
522         if (!Double.isNaN(t))
523         if (!Double.isNaN(s)) {
524           mse += (t-s)*(t-s);
525           mae += Math.abs(t-s);
526           es += s*s;
527           ms += s;
528           N++;
529         }
530       }
531       if (N > 0) {
532         mse /= N;
533         mae /= N;
534         es /= N;
535         ms /= N;
536
537         if (mse != 0.0) {
538           double psnr = 10.0 * Math.log(maxSignal*maxSignal/mse) / Math.log(10.0);
539           DecimalFormat dfn = new DecimalFormat("##0.00000000");
540           DecimalFormat dfs = new DecimalFormat("0.##E00");
541
542           String[] str = testImp.getShortTitle().split("_");
543           row[2] = str[1];
544           row[3] = (Math.abs(psnr) < 0.001 ? dfs.format(psnr) : dfn.format(psnr));
545
546         }
```

```
547         else {
548           row[2] = "Infinite";
549           row[3] = "Infinite";
550         }
551       }
552       else {
553         row[2] = "Infinite";
554         row[3] = "Infinite";
555       }
556       model.addRow(row);
557     }
558     IJ.showStatus("PSNR:" + (System.currentTimeMillis()-chrono) + " ms");
559   }
560
561   /**
562    * Private class to store a compatible ImagePlus for the
563    * computation of the SNR.
564    */
565   private class TestImage {
566     private ImagePlus imp;
567
568     public TestImage(ImagePlus imp) {
569       this.imp = imp;
570     }
571
572     public boolean isType() {
573       int type = imp.getType();
574       if (type == ImagePlus.GRAY8)
575         return true;
576       if (type == ImagePlus.GRAY16)
577         return true;
578       if (type == ImagePlus.GRAY32)
579         return true;
580       return false;
581     }
582
583     public boolean isSameSize(TestImage ref) {
584       ImagePlus imp = ref.getImagePlus();
585       if (imp.getWidth() != getImagePlus().getWidth())
586         return false;
587       if (imp.getHeight() != getImagePlus().getHeight())
588         return false;
589       return true;
590     }
591
592     public String getShortName() {
593       String title = imp.getTitle();
594       int len = title.length();
595       String html = "<html>";
596       if (len > 40)
597         html = html + title.substring(0, 35) + "..." + title.substring(len-3, len);
598       else
599         html = html + title;
600       len = title.length();
601       for (int i=len; i<42; i++)
602         title += " ";
603       title = "</html>";
604       return html;
605     }
606
607     public String getInfoName() {
```

```
608       int nx = imp.getWidth();
609       int ny = imp.getHeight();
610       return getShortName() + " (" + nx + "x" + ny + ")";
611     }
612
613     public ImagePlus getImagePlus() {
614       return imp;
615     }
616
617   }
618
619   /**
620    *  Dialog box to select the reference image.
621    */
622   private class Select extends JDialog implements ActionListener {
623     private JButton bnSelect = new JButton("Select");
624     private JButton bnCancel = new JButton("Cancel");
625     private JList list;
626     private String selection = "";
627     public Select() {
628       super(IJ.getInstance(), "Select the reference image");
629       setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
630       setLayout(new BorderLayout());
631       bnSelect.addActionListener(this);
632       bnCancel.addActionListener(this);
633       list = new JList(getDataModel());
634       JPanel panel = new JPanel(new FlowLayout());
635       panel.add(bnSelect);
636       panel.add(bnCancel);
637       add(new JLabel("List of open images"), BorderLayout.NORTH);
638       add(list, BorderLayout.CENTER);
639       add(panel, BorderLayout.SOUTH);
640       setModal(true);
641       pack();
642       GUI.center(this);
643       setVisible(true);
644     }
645
646     private DefaultListModel getDataModel() {
647       DefaultListModel model = new DefaultListModel();
648       int[] ids = WindowManager.getIDList();
649       if (ids != null) {
650         for (int i=0; i<ids.length; i++) {
651           ImagePlus imp = WindowManager.getImage(ids[i]);
652           if (imp != null) {
653             TestImage ti = new TestImage(imp);
654             if (ti.isType())
655               model.addElement(imp.getTitle());
656           }
657         }
658       }
659       return model;
660     }
661
662     public void actionPerformed(ActionEvent e) {
663       if (e.getSource() == bnSelect) {
664         int selectedIx = list.getSelectedIndex();
665         selection = (String)list.getModel().getElementAt(selectedIx);
666       }
667       dispose();
668     }
```

```
669     public String getSelected() {
670       return selection;
671     }
672 
673   }
674 
675   /**
676    * Private class of a JScrollPane with some insets.
677    */
678   private class PSNRPane extends JScrollPane {
679 
680     public PSNRPane(JTable table) {
681       super(table);
682       this.setBackground(Color.LIGHT_GRAY);
683     }
684     public Insets getInsets () {
685         return new Insets (2,2,2,2);
686     }
687   }
688 }
```

Resize_interpolation.java permet de dégrader une image en changeant sa taille en pixel avec une méthode d'interpolation.

```
1  package manga109;
2  import ij.*;
3  import ij.process.*;
4  import ij.gui.*;
5  import java.awt.*;
6 
7  import ij.plugin.PlugIn;
8  import ij.plugin.filter.*;
9  import java.io.*;
10 import java.util.GregorianCalendar;
11 import java.awt.*;
12 import java.awt.event.*;
13 import javax.swing.*;
14 import javax.swing.event.*;
15 
16 public class Resize_interpolation implements PlugInFilter, ChangeListener {
17   private ImagePlus imp;
18   private GenericDialog dialog;
19   private String[] interpolationMethods =  {"None", "Bilinear", "Bicubic"};
20   Rectangle rect;
21   private String dir, name, shortname;
22   private JSlider Width,Height;
23   private JLabel pixelw, pixelh;
24   private JSpinner spinnerw, spinnerh;
25   private int valuew, valueh;
26 
27   public int setup(String arg, ImagePlus imp) {
28     if(arg.equals("about")){
29       return DONE;
30     }
31     return DOES_ALL;
32   }
33 
34   public void run(ImageProcessor ip) {
35     rect=ip.getRoi();
36 
37     shortname=IJ.getImage().getShortTitle();
38     name= IJ.getImage().getTitle();
```

```
39      dir = IJ.getDirectory("image");
40
41      initialize();
42          if (dialog.wasCanceled()) return;
43
44      String item= dialog.getNextChoice();
45      ImageProcessor image_ip=ip.crop();
46      resizeFonction(image_ip, getInterpolation(item));
47    }
48
49    private void initialize() {
50      dialog = new GenericDialog("Insert the input");
51      Panel panel_w=new Panel();
52      Panel panel_h=new Panel();
53      Panel panel=new Panel();
54      Panel panel2=new Panel();
55      Float x=(float)1/(float)rect.width;
56      Float y=(float)1/(float)rect.height;
57
58      JLabel factor_w=new JLabel("Scale ");
59      JLabel factor_h=new JLabel("Scale ");
60
61      JLabel label_w =new JLabel("Width: ");
62      JLabel label_h =new JLabel("Height: ");
63
64      Width = new JSlider(JSlider.HORIZONTAL,1,4*rect.width, rect.width);
65      Height =new JSlider(JSlider.HORIZONTAL,1,4*rect.height, rect.height);
66      Width.addChangeListener(this);
67      Height.addChangeListener(this);
68
69      SpinnerModel w=new SpinnerNumberModel(new Float(1F), new Float(x), new Float(4F), new
            Float(0.1F));
70      spinnerw=new JSpinner(w);
71      SpinnerModel h=new SpinnerNumberModel(new Float(1F), new Float(y), new Float(4F), new
            Float(0.1F));
72      spinnerh=new JSpinner(h);
73
74      spinnerw.addChangeListener(new SpinnerChange());
75      spinnerh.addChangeListener(new SpinnerChange());
76
77      dialog.addChoice("Interpolation method: ", interpolationMethods ,
            interpolationMethods[0]);
78
79      pixelw=new JLabel(Integer.toString(Width.getValue())+"  pixelsXwidth");
80      pixelh=new JLabel(Integer.toString(Height.getValue())+"  pixelsXheight");
81
82      panel_w.add(label_w);
83      panel.add(factor_w);
84      panel.add(spinnerw);
85      //panel.add(Width);
86      panel.add(pixelw);
87
88      panel_h.add(label_h);
89      panel2.add(factor_h);
90      panel2.add(spinnerh);
91      //panel2.add(Height);
92      panel2.add(pixelh);
93
94      dialog.addPanel(panel_w);
95      dialog.addPanel(panel);
96      dialog.addPanel(panel_h);
```

```
97      dialog.addPanel(panel2);
98
99      valuew =  Width.getValue();
100     valueh = Height.getValue();
101
102     dialog.showDialog();
103   }
104
105   private int getInterpolation(String item) {
106     if (item.equals(interpolationMethods[1]))
107       return 1;
108     else if (item.equals(interpolationMethods[2]))
109       return 2;
110     return 0;
111   }
112
113   private void resizeFonction(ImageProcessor image_ip, int interpolation) {
114
115     image_ip.setInterpolationMethod(interpolation);
116
117     ImageProcessor Resize_ip=image_ip.resize(valuew, valueh);
118
119     ImagePlus imageResize=NewImage.createRGBImage("Resize image", rect.width,
            rect.height, 1, NewImage.FILL_BLACK);
120     imageResize.setProcessor("resize",Resize_ip);
121
122     ImageWindow imw=new ImageWindow(imageResize);
123     imw.setVisible(true);
124   }
125
126   public void stateChanged(ChangeEvent e){
127     Object source = e.getSource();
128     if (source==Width) {
129       valuew = Width.getValue();
130       pixelw.setText(Integer.toString(valuew) +"  pixelsXwidth");
131       spinnerw.setValue((float)valuew/rect.width);
132     }
133     else if (source==Height) {
134       valueh=Height.getValue();
135       pixelh.setText(Integer.toString(valueh) +"  pixelsXheight");
136       spinnerh.setValue((float)valueh/rect.height);
137     }
138   }
139
140   class SpinnerChange implements ChangeListener{
141     public void stateChanged(ChangeEvent e) {
142       Object source = e.getSource();
143       if (source==spinnerw){
144         valuew=(int)((Float.parseFloat(spinnerw.getValue().toString()))*rect.width);
145         Width.setValue(valuew);
146       }
147       else if(source == spinnerh){
148         valueh=(int)((Float.parseFloat(spinnerh.getValue().toString()))*rect.height);
149         Height.setValue(valueh);
150       }
151     }
152   }
153
154 }
```

Webographie

[WWW1] DUSSON Alexandre BENOIT Christophe. *TIPE sur la compression de données informatiques*. URL : http://mp01.free.fr/comp/comp.htm (visité le 2016).

[WWW2] Matthieu Roger Dau-Khoî Nguyen. *Peut-on detecter qu'une image a subi un codage JPEG*. URL : http://www.tsi.telecom-paristech.fr/pages/enseignement/ressources/beti/test_JPEG/mti.html (visité le 2016).

[WWW3] ImageJ. *ImageJ ij package*. URL : https://imagej.nih.gov/ij/developer/api/ij/plugin/filter/package-summary.html (visité le 2016).

[WWW4] *Interpolation bilinéaire*. URL : http://www.f-legrand.fr/scidoc/docimg/image/niveaux/interpolation/interpolation.html (visité le 2016).

[WWW5] *L'algorithme JPEG*. URL : http://www-ljk.imag.fr/membres/Valerie.Perrier/SiteWeb/node6.html (visité le 2016).

[WWW6] *Traitement d'images Introduction*. URL : http://dept-info.labri.fr/~vialard/Traitement/cours/cours1.pdf (visité le 2016).

[WWW7] Wikipédia. *Interpolation bilinéaire*. URL : https://fr.wikipedia.org/wiki/Interpolation_bilin%C3%83%C2%A9aire (visité le 2016).

Bibliographie

[1] ImageJ/Fiji 1.46. *ImageJ User Guide IJ 1.46r Revised edition*. Rapp. tech. 2015.

[2] S. Hadjihassan A. Zergaïnoh et J.-P. Astruc. *Interpolation statistique à partir d'un échantillonnage irrégulier pour la reconstruction d'images*. Rapp. tech. Laboratoire de Traitement et de Transport de l'Information, Université de Paris Nord, Institut Galilée, 1999.

[3] Ken Cabeen et Peter Gent. *Image compression and the discrete consine transform*. Rapp. tech.

[4] Frédéric Dufaux. *Compression d'images*. Rapp. tech. Département Traitement du Signal et des Images TELECOM ParisTech, 2011.

[5] Anne Marie Mortier. *La question de redimensionnement*. Rapp. tech. Université lyon 2, 2005.

Méthode de dégradation image par re-compression JPEG et re-échantillonnage

Résumé

Dans ce projet, on s'intéressera à fabriquer les duplicatas des images de référence par une dégradation des images sur deux niveaux : re-compression et re-échantillonnage

Mots-clés

compression, re-échantillonnage, JPEG, facteur de qualité, DPI, PSNR, interpolation, ImageJ

Abstract

In this project, we are interested to make duplicates of the reference images in two levels: re-compression and re-sampling

Keywords

compression, re-sampling, JPEG, quality factor, DPI, PSNR, interpolation, ImageJ

Tuteurs académiques
Mathieu Delalandre
Sebastien Aupetit

Étudiants
ZHANG Shimeng (DI5)