Filtres d'image

Préambule

Ce TP utilise la bibliothèque Pillow.

Nous allons travailler à partir de deux photos mises à disposition par Hans Stieglitz sur les Wikimedia commons, et soumise à la licence CC-BY-SA 3.0 :

- tigre.jpg
- tigrenb.png

Codage des couleurs

Il existe plusieurs façons de coder les couleurs d'une image. Nous en présentons ici deux : le système RVB et le système CMJN. Le système CMJN est utilisé pour l'impression, tandis que le système RVB est utilisé pour la lumière (écran, projecteurs, ...).

Le système RVB

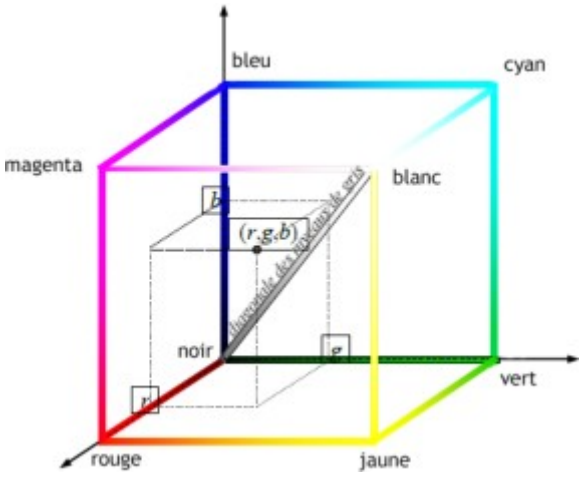
Il existe plusieurs façons de décrire les couleurs en informatique. Nous présentons ici une des plus utilisées : le codage RVB, qui est utilisé notamment dans les formats d'image JPEG et TIFF. Rouge vert bleu, abrégé RVB (ou RGB de l'anglais red, green, blue), est un format de codage des couleurs. Ces trois couleurs sont les couleurs primaires en synthèse additive. Elles correspondent en fait à peu près aux trois longueurs d'ondes auxquelles répondent les trois types de cônes de l'œil humain (voir trichromie). L'addition des trois donne du blanc pour l'œil humain. Elles sont utilisées en éclairage afin d'obtenir toutes les couleurs visibles par l'homme. Elles sont aujourd'hui utilisées en vidéo, pour l'affichage sur les écrans, et dans les logiciels d'imagerie.

C'est sur ce principe que fonctionnent les téléviseurs couleur. Si vous regardez un écran de télévision couleur avec une loupe, vous allez voir apparaître des groupes de trois points lumineux : un rouge, un vert et un bleu. La combinaison de ces trois points donne un point lumineux (un pixel) d'une certaine couleur.

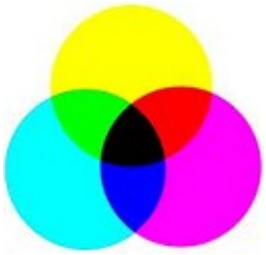
Le système RVB est une des façons de décrire une couleur en informatique. Ainsi le triplet {255, 255, 255} donnera du blanc, {255, 0, 0} un rouge pur, {100, 100, 100} un gris, etc. Le premier nombre donne la composante rouge, le deuxième la composante verte et le dernier la composante bleue.

Le cube des couleurs

On peut représenter chacune de ces couleurs comme un point d'un cube de l'espace de dimension trois en considérant un repère orthonormé dont les trois axes r, g, b représentent les intensités de rouge, de vert et de bleu. L'origine représente ainsi le noir (r=g=b=0) et le point opposé (r=g=b=255) le blanc. Les trois sommets (255,0,0), (0,255,0) et (0,0,255) représentent les trois couleurs de base (rouge, vert, bleu) et les trois sommets opposés, (0,255,255), (255,0,255) et (255,255,0), le cyan, le magenta et le jaune. La grande diagonale de ce cube joignant le noir et le blanc est l'axe achromatique, i.e. l'axe des niveaux de gris.



Le système CMJN

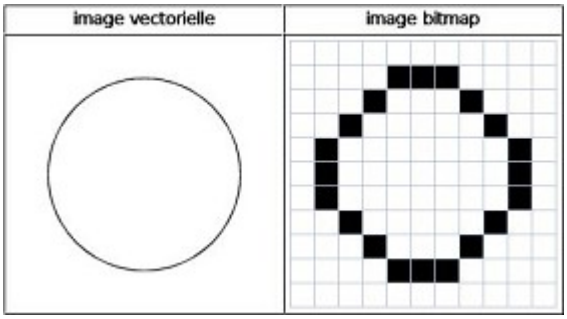


La quadrichromie ou CMJN (cyan, magenta, jaune, noir ; en anglais CMYK, cyan, magenta, yellow, key) est un procédé d'imprimerie permettant de reproduire un large spectre colorimétrique à partir des trois teintes de base (le cyan, le magenta et le jaune ou yellow en anglais) auxquelles on ajoute le noir (key en anglais). L'absence de ces trois composantes donne du blanc tandis que la somme des trois donne du noir. Toutefois, le noir obtenu par l'ajout des trois couleurs Cyan, Magenta et Jaune n'étant que partiellement noir en pratique (et coûtant cher), les imprimeurs rajoutent une composante d'encre noire.

Formats d'images

On désigne sous le terme d'image numérique toute image acquise, créée, traitée ou stockée sous forme binaire (suite de 0 et de 1).

Images matricielles (ou images bitmap)



Elles sont composées, comme leur nom l'indique, d'une matrice (tableau) de points colorés. Dans le cas des images à deux dimensions (le plus courant), les points sont appelés pixels. Ce type d'image s'adapte bien à l'affichage sur écran informatique ; il est en revanche peu adapté pour l'impression, car la résolution des écrans informatiques, généralement de 72 à 96 ppp (« points par pouce », en anglais dots per inch ou dpi) est bien inférieure à celle atteinte par les imprimantes, au moins 600 ppp aujourd'hui. L'image imprimée, si elle n'a pas une haute résolution, sera donc plus ou moins floue ou laissera apparaître des pixels carrés visibles.

Les formats d'images matricielles les plus courants sont jpeg, gif, png, tiff, bmp.

Définition et résolution

La définition d'une image matricielle est donnée par le nombre de points la composant. En image numérique, cela correspond au nombre de pixels qui composent l'image en hauteur (axe vertical) et en largeur (axe horizontal) : 200 pixels x 450 pixels par exemple.

La résolution d'une image matricielle est donnée par un nombre de pixels par unité de longueur (classiquement en ppp). Ce paramètre est défini lors de la numérisation (passage de l'image sous forme

binaire), et dépend principalement des caractéristiques du matériel utilisé lors de la numérisation. Plus le nombre de pixels par unité de longueur de la structure à numériser est élevé, plus la quantité d'information qui décrit cette structure est importante et plus la résolution est élevée. La résolution d'une image numérique définit donc le degré de détail de l'image. Ainsi, plus la résolution est élevée, meilleure est la restitution. Cependant, pour une même dimension d'image, plus la résolution est élevée, plus le nombre de pixels composant l'image est grand. Le nombre de pixels est proportionnel au carré de la résolution, étant donné le caractère bidimensionnel de l'image : si la résolution est multipliée par deux, le nombre de pixels est multiplié par quatre. Augmenter la résolution peut entraîner des temps de visualisation et d'impression plus longs, et conduire à une taille trop importante du fichier contenant l'image et à de la place excessive occupée en mémoire.

Filtres d'image

Cette partie du TP concerne l'algorithmique de l'image. Plus précisément, on manipulera des images matricielles, c'est-à-dire représentées par des tableaux de pixels.

On utilise Pillow pour s'affranchir de la question des formats de fichiers.

Ouverture et enregistrement de fichiers d'image avec Pillow

Le bout de code suivant convertit le fichier tigre.jpg (au format JPEG) en tigre.png (au format PNG) :

```
import PIL.Image as Image
img = Image.open(r'tigre.jpg')
img.save(r'tigre.png')
```

Informations sur une image

Dans un interpréteur Python, essayez :

```
import PIL.Image as Image
img = Image.open(r'tigre.jpg')
```

puis essayez d'utiliser la documentation interne de python (par exemple via la fonction dir de Python ou la commande help de l'interpréteur) pour explorer les informations fournies par l'objet image.

Essayez par exemple d'obtenir sa taille.

Représentation d'une image en mémoire

Si img est une image chargée avec PIL.Image.open, on accède à ses pixels via la méthode img.load() qui renvoie un tableau indexé par des couples d'entiers (et non pas une matrice au sens python du terme).

Par exemple :

```
pixels = img.load()
print(pixels[0,0])
```

renvoie la valeur du pixel en haut à gauche de l'image.

Affichez des pixels de l'image en couleurs, puis de l'image en noir et blanc. Conclure.

Modifier une image

Pour modifier un pixel, on change sa valeur dans le tableau des pixels :

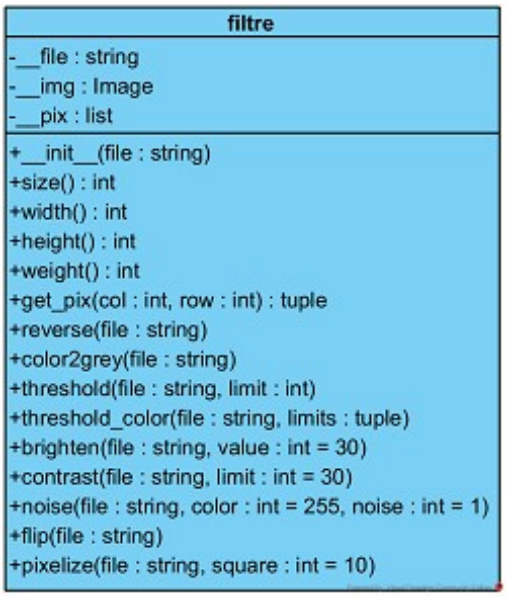
```
pixels[0,0] = 0
```

```
img.save(r'tigre_mod.png')
```

Est-ce que ça fonctionne avec l'image en noir et blanc ? Avec celle en couleurs ? Conclure.

Premiers filtres

Vous êtes parés pour écrire votre premier filtre.



1. Créer un fichier python filtre.py.
2. Écrire une classe filtre à partir du diagramme de classe ci-contre.

   NB : le corps des méthodes ne sera pas développé immédiatement ; on utilisera l'instruction Python **pass** en attendant.
3. Compléter le constructeur de classe.
4. Compléter les méthodes suivantes :
   - size()

     retourne la taille en pixels d'une image sous forme de tuple largeur, hauteur
   - width()

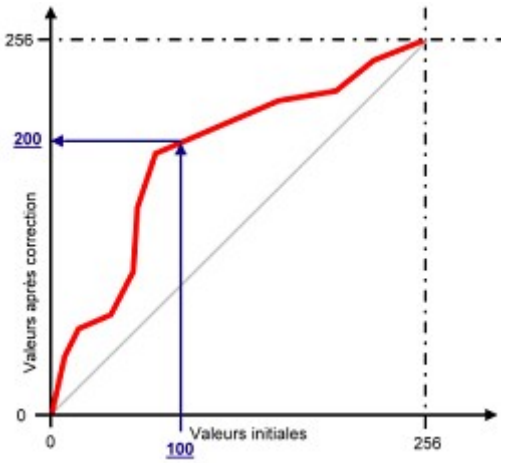
     retourne la largeur d'une image en pixels
   - height()

     retourne la hauteur d'une image en pixels
   - weight()

     retourne le poids d'une image en pixels
   - get_pix(x, y)

     retourne la valeur du pixel de coordonnées (x,y), ou None si erreur

Retoucher une image revient à modifier les valeurs de certains pixels. On peut le faire localement (à un endroit bien précis de l'image) ou globalement. Dans ce dernier cas, on utilise un outil appelé « courbe tonale », qui ressemble au dessin ci-contre.



Sur l'abscisse, on lit les valeurs originales des pixels et sur l'ordonnée les valeurs après modifications. Sur le graphique ci-contre, tous les pixels de valeurs 100 prendront la valeur 200. Ils vont donc s'éclaircir. La diagonale grise est la courbe où il n'y a aucune modification.

En fait, il y a trois courbes tonales : une pour le rouge, une pour le vert et une pour le bleu. On les modifie souvent simultanément de la même façon, mais on peut aussi les modifier séparément.

Négatif

Écrire une méthode reverse() qui remplace tous les pixels de l'image par leur valeur en négatif. Obtenir le négatif d'une image est très simple : toutes les composantes x de tous les pixels de l'image sont remplacées par 255 – x.

Rouge

Chaque pixel de l'image est une combinaison de rouge, de vert et de bleu. En assignant la valeur 0 aux composantes verte et bleue, on obtient une image à dominante rouge.

Écrire une méthode red() qui réalise cette opération.

Niveaux de gris

Dans une image en niveaux de gris, chaque pixel est noir, blanc, ou a un niveau de gris entre les deux. Cela signifie que les trois composantes ont la même valeur.

L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. Pour simplifier les choses, nous prendrons ici la moyenne « classique ».

Écrire une méthode color2grey() qui transforme une image en couleurs vers une image en niveaux de gris.

NB : créer la nouvelle image en utilisant Image.new en nivaux de gris (mode "L"). Voir help(Image.new) dans l'interpréteur.

Seuillage

Le seuillage d'image est la méthode la plus simple de segmentation d'image.

À partir d'une image en niveau de gris, le seuillage d'image peut être utilisé pour créer une image comportant uniquement deux valeurs, noir ou blanc (monochrome). On remplace un à un les pixels d'une image par rapport à une valeur seuil fixée (par exemple 50). Ainsi, si un pixel à une valeur supérieure au seuil, il prendra la valeur 255 (blanc), et si sa valeur est inférieure, il prendra la valeur 0 (noir).

Avec une image en couleur, on fera de même avec les trois composantes rouge, vert et bleu. Il y aura ainsi huit couleurs possibles pour chaque pixel : blanc, noir, rouge, vert, bleu, magenta, jaune et cyan.

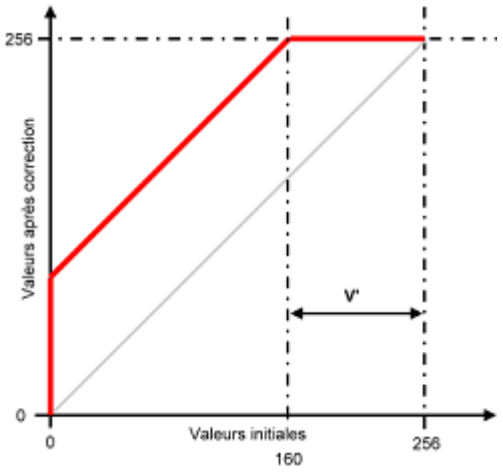
Écrire les méthodes threshold() et threshold_color() qui réalisent un seuillage pour un seuil donné en paramètre.

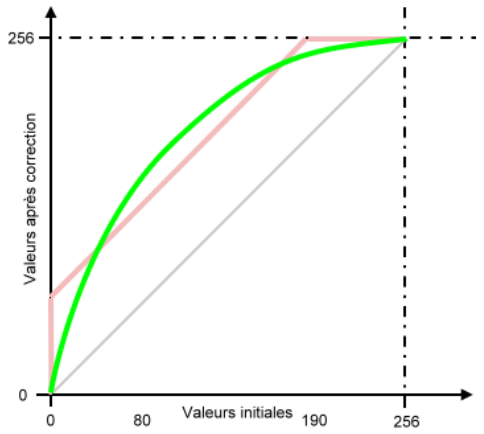
Luminosité

Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe à tous les niveaux.

Pour une valeur de + 96, tous les points de l'espace V' seront blancs.

- Première conséquence : les points les plus noirs auront une valeur égale à 96 et il n'existera plus aucun point entre 0 et 96.
- Deuxième conséquence : les points ayant une valeur supérieure à 160 deviendront des points parfaitement blancs, puisque la valeur maximale possible est 255. Il y a donc perte d'informations.



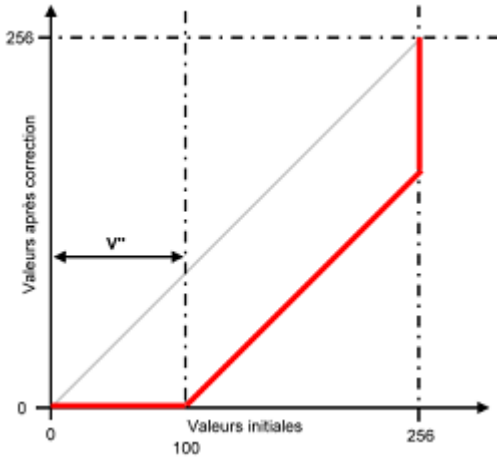


Pour éviter ces pertes d'informations, il faut que la courbe tonale rejoigne les axes tangentiellement, comme dans l'exemple ci-contre. Ainsi, aucun point de débordera des valeurs limites minimale (0) ou maximale (255). Il sera en particulier possible de revenir en arrière.

Pour diminuer la luminosité il faudra au contraire soustraire une valeur fixe à tous les niveaux.

Pour une valeur de -100, tous les points de l'espace V" seront noirs.

- Première conséquence : les points les plus blancs auront une valeur égale à 156 et il n'existera plus aucun point entre 156 et 255.
- Deuxième conséquence : les points ayant une valeur comprise entre 0 et 100 deviendront noirs, puisque la valeur minimale est 0. Il y aura donc là aussi perte d'informations.

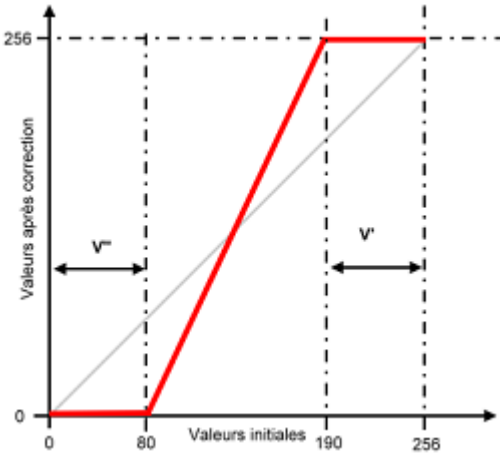


Écrire une méthode brighten() qui réalisent un éclaircissement de l'image pour une valeur donnée en paramètre.

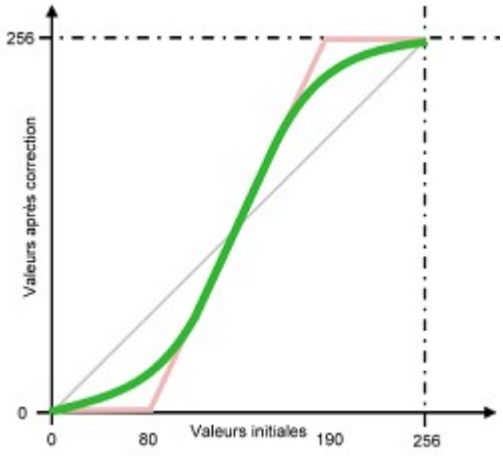
Contraste

Pour rendre une image plus contrastée, il faut assombrir les points foncés et éclaircir les points clairs, par exemple comme dans la figure ci-contre.

Les points de l'espace V" seront noirs et ceux de l'espace V' blancs.



Pour les mêmes raisons que précédemment, cette manière de faire va causer des pertes d'informations. Aussi faut-il adoucir la courbe.



Écrire une méthode contrast() qui effectue un contraste de l'image en fonction d'une valeur donnée en paramètre.

Bruit

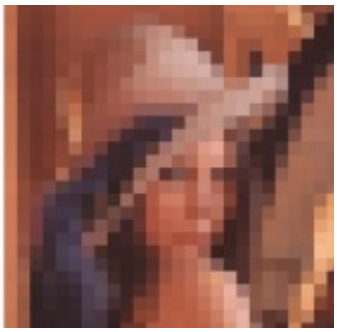
Le « bruit » consiste à remplacer aléatoirement un certains nombre de pixels par des pixels blancs.

Écrire une méthode noise() qui prend en paramètres la couleur du bruit (valeur de 0 à 255) et la fréquence d'apparition (0 à 10).

Symétrie axiale d'axe horizontal

La symétrie axiale horizontale consiste à échanger les pixels du haut de l'image avec ceux du bas. Ainsi, chaque pixel de la rangée 0 sera échangé avec celui en dessous de lui à la rangée filtre.height() - 1 ; ceux de la rangée 1 avec ceux de la rangée filtre.height() - 2, etc.

Pixellisation (pour les plus rapides)



L'image est divisée en carrés de la taille spécifiée (exemple : 10x10 pixels). Chaque carré est ensuite rempli avec la couleur moyenne de la zone.

Écrire une méthode pixelize() qui prend en paramètre la taille du carré en pixels.

1. Créer un fichier filtre_main.py.
2. Lancer toutes les méthodes de l'objet filtre.

Résultats attendus :


tigre_bright.jpg


tigre_contrast.jpg


tigre_grey.jpg


tigre_noise.jpg


tigre_pixel.jpg


tigre_rev.jpg


tigre_threshold.jpg


tigre_threshold_bw.jpg


tigre_red.jpg