TD 14 — IMAGES

1 Représentation informatique des images

Une image est un tableau (bidimensionnel) de pixels. Chaque pixel a une couleur, qui est en général codée :

- soit par une entier entre 0 et 255 (dans ce cas, l'image est en niveaux de gris, le 0 correspond à un pixel noir et le 255 à un pixel blanc) ;
- soit par un triplet (r, g, b) d'entiers compris entre 0 et 255. Ici, le r correspond à la composant rouge, le g à la composante verte (*green*) et le b à la composante bleue. La couleur est obtenue par *synthèse additive* : $(0, 0, 0)$ correspond au noir, $(200, 200, 200)$ à un gris clair, $(255, 0, 0)$ à un rouge éclatant, $(255, 255, 0)$ à un jaune vif (rouge + vert), $(100, 50, 150)$ à un violet assez foncé…

2 Le module PIL de Python

Nous allons utiliser le module `PIL`, ou plutôt un petit morceau de ce module. `PIL` (Python Imaging Library) ne fait pas partie de la bibliothèque standard de Python : il s'agit d'une bibliothèque supplémentaire (libre) qu'il faut installer.

Pour les fonctions les plus élémentaires, on peut regarder l'exemple suivant :

```
from PIL import Image

im = Image.open("chat.jpg")
# charge le fichier "chat.jpg" (qui doit se trouver dans le même
# répertoire que notre fichier python) dans l'objet im

im.show()
# affiche l'image dans une fenêtre graphique

coin_h_g = im.getpixel((0, 0))
# récupère la valeur du pixel en haut à gauche (sous la forme d'un
# triplet (r, g, b)).

largeur, hauteur = im.size
# dimensions de l'image (en nombre de pixels)

im.putpixel((0, hauteur - 1)) = (255, 0, 0)
# colorie le pixel en bas à gauche en rouge

im.save("stage_modifie.jpg")
```

On peut déjà signaler deux méthodes souvent plus pratiques pour accéder aux pixels d'une image (que l'on supposera avoir été ouverte dans l'objet `im` par la commande `im = Image.open(nom_de_fichier)`) :

- on peut récupérer un tableau contenant les pixels par `pixels = im.load()`. On peut alors accéder au pixel de coordonnées (x, y) par `pixels[x, y]` (que ce soit pour lire ou écrire) ;

- on peut aussi récupérer une liste «plate» contenant tous les pixels par `pixels = im.getdata()`. Ce n'est pas très pratique si l'on souhaite raisonner sur la géométrie de l'image, mais cela peut l'être lorsque l'on souhaite appliquer le même traitement à tous les pixels de l'image.

Enfin, si l'on souhaite créer une nouvelle image (initialement noire), on utilisera la fonction `Image.new('RGB', (largeur, hauteur)`. On peut remplacer `'RGB'` par `'L'` si l'on souhaite créer une image en niveaux de gris.

3 Quelques exemples de manipulations

On partira d'une image hautement scientifique que vous pouvez récupérer sur mon site (`http://bianquis.org`) :

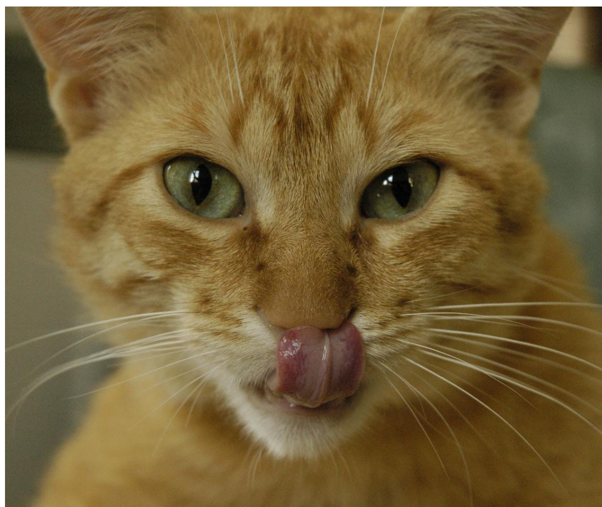


Figure 14.1 – L'image initiale.

3.1 Transformations géométriques

Exercice 1

Écrire une fonction `miroir(image)` *qui renvoie une nouvelle image, "miroir" de l'argument que l'on a donné :*

```
chat_miroir = miroir(chat)
chat_miroir.show()
```

Exercice 2

Écrire une fonction `mosaique(image)` *donnant le résultat suivant :*

```
chat_abime = mosaique(chat)
chat_abime.show()
```

3.2 Contraste et luminosité

Exercice 3

On souhaite convertir une image couleur en niveaux de gris. Pour cela, on créera une nouvelle image avec la commande `Image.new('L', (largeur, hauteur))`. *Dans ce cas, chacun des pixels ne sera constitué que d'une valeur entière entre* 0 *et* 255. *Le plus «naturel» est de prendre la moyenne des trois valeurs r, g et b (qu'on appelle* luminance*), mais ce n'est pas toujours ce qui donne le meilleur résultat. On va donc écrire une fonction* `gris(im, coef_r, coef_g, coef_b)` *qui renvoie une version en niveau de gris de* `im` *calculée en prenant la moyenne coefficientée des composantes r, g et b.*

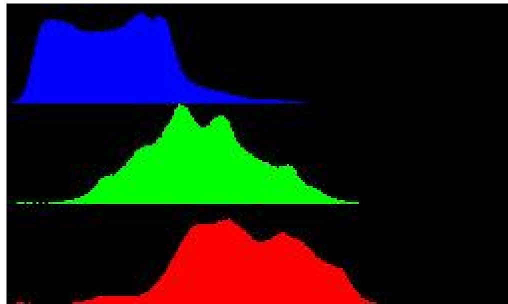
```
gris(chat, -1, 5, 3).show()
```

Exercice 4

*Un outil utile (pour la retouche de photographie en particulier) est l'*histogramme *d'une image. Essentiellement, il s'agit de compter, pour chaque valeur possible d'un pixel, le nombre de pixels de l'image qui ont cette valeur. On peut obtenir soit trois histogrammes (un par couleur), soit un seul si l'on s'intéresse à la luminance (valeur moyenne rgb des pixels).*

1. *Écrire une fonction* `histogramme(im)` *qui renvoie un triplet de listes* `(h_r, h_g, h_b)` *de longueur* 256 *telle que* `h_r[i]` *soit égal au nombre de pixels de l'image dont la valeur «rouge» vaut i (de même pour* `h_g` *et le vert...).*

2. *Écrire une fonction* `affiche_histo(h)` *qui prenne un triplet tel que celui produit par la fonction précédente et affiche un graphique :*



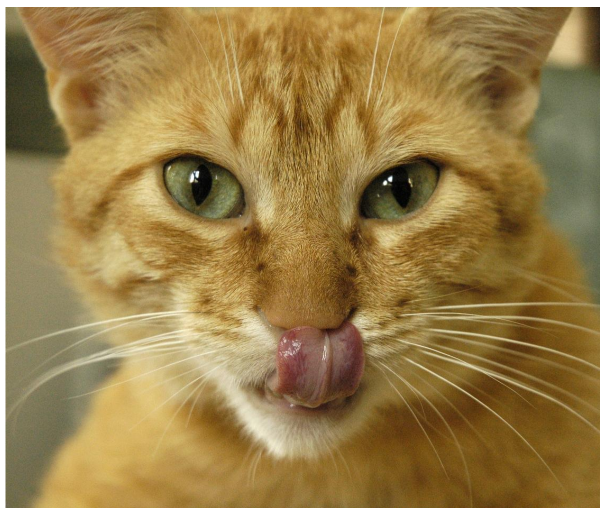
3. *Une manière courante d'augmenter le contraste d'une image est «d'étendre» l'histogramme : dans l'exemple ci-dessus, on voit qu'il n'y a quasiment pas de pixels de couleur très clair, une transformation affine sur les valeurs des pixels permettrait donc de mieux utiliser l'espace des couleurs disponibles.*

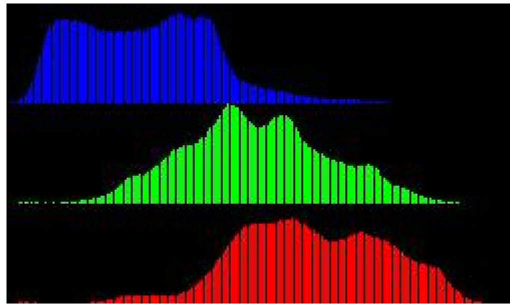
a. *Écrire une fonction* `ecrete(x)` *qui renvoie* 0 *si* $x < 0$, $\lfloor x \rfloor$ *si* $0 \leqslant x < 256$ *et* 255 *si* $x \geqslant 256$.

b. *Écrire une fonction* `min_max_seuil(histos, seuil)` *qui prend un triplet d'histogrammes* (h_r, h_g, h_b) *et un entier* `seuil` *et renvoie le couple* (m, M) *où m est le plus petit indice i tel que l'un au moins des trois histogrammes vaille au moins* `seuil` *pour cet indice, et M le plus grand de ces indices.*

c. *Écrire alors une fonction* `optimise_contraste(im)` *appliquant aux valeurs des pixels de* `im` *une transformation affine qui envoie m sur* 0 *et M sur* 255 *(avec le m et le M définis à la question précédente).*

On obtient l'image suivante (dont on a également mis le nouvel histogramme) :





3.3 Couleurs

Exercice 5

Écrire une fonction qui inverse les canaux vert et rouge d'une image.

Exercice 6

Écrire une fonction `negatif(im)` *qui applique la transformation $x \mapsto 255 - x$ à chacun des canaux de l'image.*

3.4 Flou et accentuation

Pour flouter une image, ou au contraire en accentuer les détails, la technique la plus simple consiste à remplacer la valeur de chaque pixel par une valeur calculée sur une petite fenêtre (3×3 typiquement) centrée autour de ce pixel. Ce calcul est basée sur une matrice qui varie suivant l'effet que l'on veut obtenir.

Par exemple, en prenant $M = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$, on remplacera la valeur r d'un pixel par la moyenne coefficientée des valeurs r pour les 9 pixels alentour, affectées des coefficients donnés par la matrice (donc 10 pour le pixel lui-même, par exemple).

Écrire une fonction `applique_filtre(im, filtre)` qui prend en entrée une image et un filtre sous la forme d'une matrice 3×3 et effectue l'opération décrite ci-dessus. On pourra essayer avec la matrice M (qui produira un flou assez léger) et avec $M' = \begin{pmatrix} -1 & -2 & -1 \\ -2 & 16 & -2 \\ -1 & -2 & -1 \end{pmatrix}$, qui produira une accentuation assez marquée des détails.

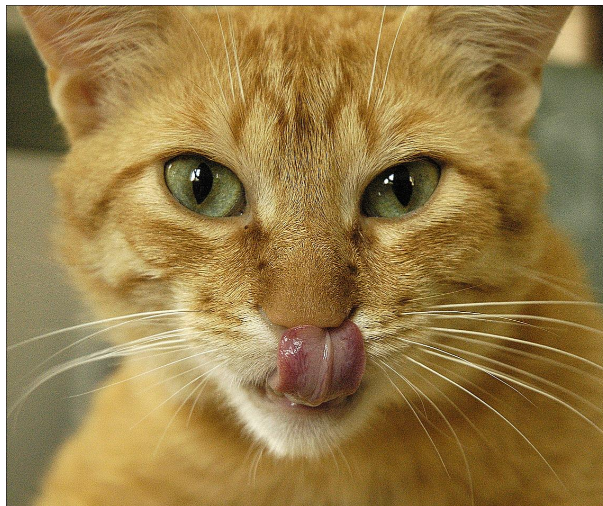


Figure 14.2 – L'image après optimisation du contraste et accentuation des détails.