Chapitre 4

Traitement d'images

4.1. Codage des couleurs

Il existe plusieurs façons de coder les couleurs. 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, ...).

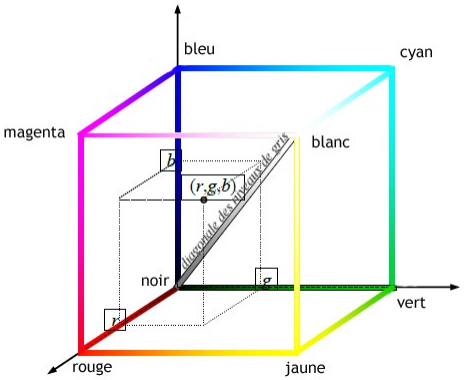
4.1.1. Le système RVB

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. Les trois composantes 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. 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 : en vidéo, pour l'affichage sur les écrans, et dans les logiciels d'imagerie.

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.

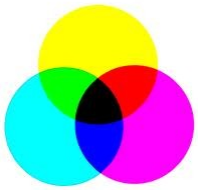
4.1.2. Le cube des couleurs



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.

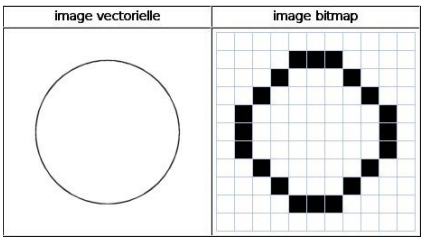
4.1.3. Le système CMJN



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.

4.2. 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).



4.2.1. 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 jpg, 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.

Le format JPG (ou jpeg)

Le format le plus communément utilisé pour toute image standard est le JPG. C'est un format qui permet de compresser considérablement les images sans trop les altérer.

Le format PNG

Dans le format PNG, chaque pixel est codé avec trois nombres réels compris entre 0 et 1 représentant les composantes rouge, verte et bleue.

Le format PNG est non destructeur, c'est-à-dire qu'il ne provoque pas de dégradation de l'image, et offre 9 niveaux de compression à l'image. Néanmoins, on l'utilise généralement lorsque l'on souhaite avoir des transparences dans l'image.

Le format GIF

Le format GIF, quant à lui, possède les mêmes propriétés que le PNG, dans la mesure où il accepte les transparences. En revanche, il est beaucoup plus destructeur puisque sa palette de couleur se limite à 256 tonalités, alors que le PNG peut en gérer jusqu'à 16 millions. L'avantage du format GIF est que l'on peut créer des images animées.

Dans le format GIF, chaque pixel est codé sur 4 octets, avec les trois premières valeurs représentant les composantes rouge, verte et bleue, et la quatrième valeur représentant le canal alpha.

Le canal alpha, ou couche alpha, permet de gérer la transparence du pixel. Il prend une valeur comprise entre 0 et 255 : une valeur de 0 signifie que le pixel est entièrement transparent, une valeur de 255 signifie que le pixel est entièrement opaque. Les valeurs intermédiaires permettent des niveaux de transparence semi-opaques.

Le format WebP

Le format WebP a été développé par Google en 2010, et il possède un excellent rapport qualité/compression. De plus, il prend en charge la transparence, les images animées et les images à plage dynamique élevée. Il utilise un algorithme de compression basé sur la prédiction qui exploite les similitudes entre les pixels adjacents pour compresser les données de l'image.

Au final, le format WebP est généralement plus petit que les formats d'image traditionnels tels que JPEG et PNG, ce qui peut entraîner des temps de chargement de page plus rapides.

4.2.2. Images vectorielles

Le principe des images vectorielles est de représenter les données de l'image par des formules géométriques qui vont pouvoir être décrites d'un point de vue mathématique. Cela signifie qu'au lieu de mémoriser une mosaïque de points élémentaires, on stocke la succession d'opérations conduisant au tracé. Par exemple, un dessin peut être mémorisé par l'ordinateur comme « une droite tracée entre les points (x_1, y_1) et (x_2, y_2) », puis « un cercle tracé de centre (x_3, y_3) et de rayon 30 de couleur rouge ». C'est le processeur qui sera chargé de traduire ces formes en informations interprétables par la carte graphique.

L'avantage de ce type d'image est la possibilité de l'agrandir indéfiniment sans perdre la qualité initiale, ainsi qu'un faible encombrement.

L'usage de prédilection des images vectorielles concerne les schémas générés avec certains

logiciels de DAO (Dessin Assisté par Ordinateur) comme *AutoCAD*.

Étant donné que les moyens de visualisation d'images actuels comme les moniteurs d'ordinateur reposent essentiellement sur des images matricielles, les descriptions vectorielles (Fichiers) doivent préalablement être converties en descriptions matricielles avant d'être affichées comme images.

Quelques formats d'images vectorielles : ai (Adobe Illustrator), eps (encapsulated postscript), pdf (portable document format), svg (scalable vector graphics), swf (flash).

4.3. Manipuler des images matricielles avec Python

Dans la suite de ce chapitre, nous allons travailler avec des images aux formats GIF et PNG, deux formats qui se ressemblent beaucoup. Pour simplifier un peu les choses, toutes les images seront en couleurs.

La structure de données Python typiquement utilisée pour représenter des tableaux de pixels est l'`array` du module numpy. Les valeurs sont soit des entiers de 0 à 255 (format gif), soit des réels de 0.0 à 1.0 (format png).

Affichage facile

Pour afficher les images, nous utiliserons la fonction `display(img_list, size, shape)` ci-dessous, qui affiche une série d'images stockées dans une liste `img_list`. La taille de l'affichage d'une image est fixée à `size` pouces. Le booléen `shape` permet d'afficher ou non les dimensions du tableau au dessus de l'image.

Fonction tirée de [2].

```
import numpy as np
import matplotlib.pyplot as plt

def display(imglist, size=5, shape=True):
    cols = len(imglist)
    fig = plt.figure(figsize=(size*cols,size*cols))
    for i in range(0,cols):
        a = fig.add_subplot(1, cols, i+1)
        subfig = plt.imshow(imglist[i], vmin=0.0, vmax=1.0)
        subfig.axes.get_xaxis().set_visible(False)
        subfig.axes.get_yaxis().set_visible(False)
        if shape == True:
            a.set_title(str(imglist[i].shape))
    plt.show()
```

Dans nos images de hauteur h et de largeur w, le pixel de coordonnées (0 ; 0) se trouve dans le coin supérieur gauche tandis que le pixel du coin inférieur droit a pour coordonnées (h–1 ; w–1). La hauteur h est donc le nombre de lignes, tandis que la largeur w est le nombre de colonnes.

Pour accéder au pixel situé à la i-ème ligne et à la j-ème colonne de l'image on utilise donc simplement `img[i,j]`.

Par exemple, le pixel situé sur la ligne 78 et la colonne 95 de l'image (au format png) correspond au vecteur :

```
p = img[78,95]
print(p)
```

```
[0.7411765, 0.6627451, 0.5803922]
```

Les composantes R, G et B de ce pixel `p` sont :

```
print("Rouge: ", p[0])
print("Vert : ", p[1])
print("Bleu : ", p[2])
```

```
Rouge:  0.7411765
Vert :  0.6627451
Bleu :  0.5803922
```

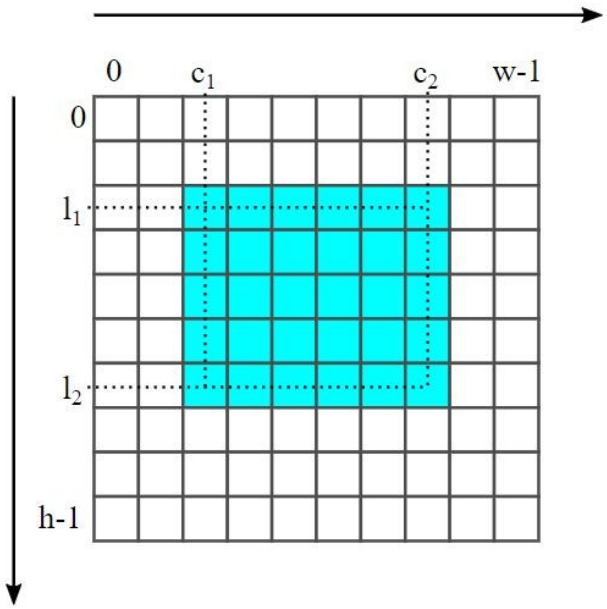
On peut accéder directement à une composante d'un pixel en spécifiant son indice :

```
img[78,95,0]
```

```
0.7411765
```

Exemples d'utilisation

Le code Python ci-dessous permet d'extraire une zone rectangulaire d'une image, située entre les lignes l_1 et l_2, et les colonnes c_1 et c_2 :




```
# Extrait d'une image

def rectangle(img,l1,c1,l2,c2):
    pixels = []
    for i in range(l1,l2):      # hauteur
        ligne = []
        for j in range(c1,c2):  # largeur
            p = img[i,j]
            ligne.append(p)
        pixels.append(ligne)
    return np.array(pixels)

img = plt.imread("images/mandrill.gif")     # image au format gif
#img = plt.imread("images/solo-256px.png")    # image au format png
img2 = rectangle(img,50,100,90,200)
img3 = rectangle(img,100,100,200,210)
display([img,img2,img3],4)
```

Résultat :

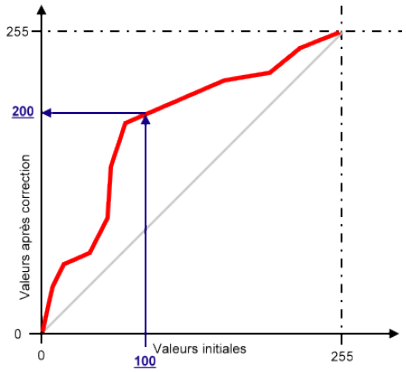
Le code Python ci-dessous permet de calculer le négatif d'une image, pour les formats gif et png. Obtenir le négatif d'une image au format gif est très simple : les trois composantes x de tous les pixels de l'image sont remplacées par $255–x$ ($1–x$ pour le format png). Le canal alpha reste inchangé.


```
# Négatif d'une image

def negatif(img,type):
    # renvoie le négatif d'une image
    img2 = img.copy()                   # On copie d'abord l'image à modifier
    for i in range(img.shape[0]):       # hauteur
        for j in range(img.shape[1]):   # largeur
            p = img[i,j]                # pixel de coordonnées (i,j)
            if type == "gif":
                img2[i,j] = [255 - p[0], 255 - p[1], 255 - p[2], 255]
            else:  # format png
                img2[i,j] = [1.0 - p[0], 1.0 - p[1], 1.0 - p[2]]
    return img2


img = plt.imread("images/mandrill.gif")     # une image au format gif
img2 = negatif(img, "gif")
display([img,img2],4)
```

Et voici le résultat :

4.4. Courbe tonale

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. La diagonale grise est la courbe où il n'y a aucune modification.

Il y a en fait 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.

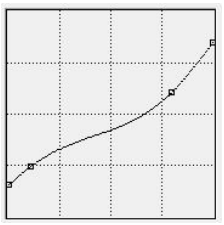
La courbe tonale permet d'ajuster avec une grande souplesse l'intensité des couleurs de l'image à numériser. Vous pouvez ajuster les valeurs d'ombre, de haute lumière et les tons intermédiaires et obtenir ainsi un bon équilibre des couleurs de l'image.

Exercice 4.1

Associez chacun des chats ci-dessous à sa courbe tonale.
L'image ci-contre est l'image originale, avant traitement.

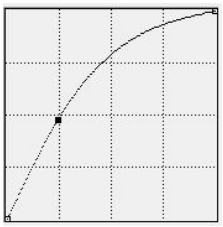
A

1



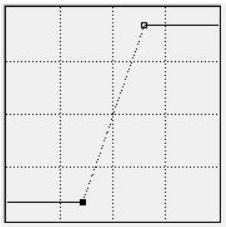
B

2



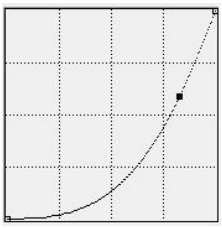
C

3



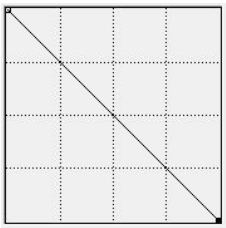
D

4



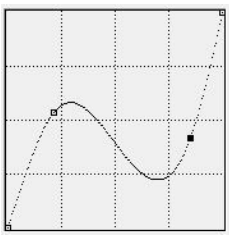
E

5



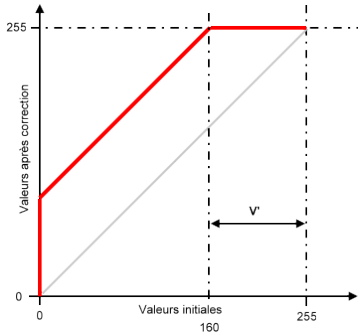
F

6



4.4.1. Changer la luminosité

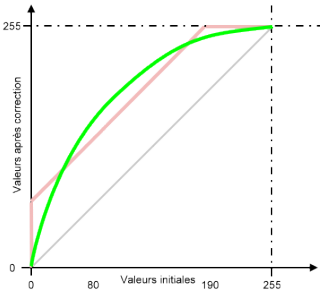
Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe à toutes les composantes.



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

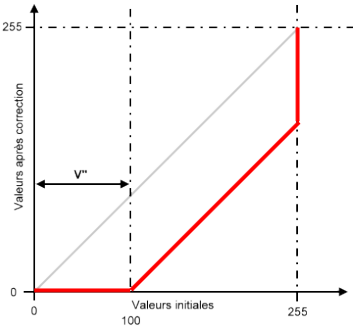
Première conséquence : les points les plus foncés auront une valeur égale à 96 et il n'existera plus aucun point avec une valeur comprise entre 0 et 96.

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 clairs auront une valeur égale à 156 et il n'existera plus aucun point avec une valeur comprise entre 156 et 255.

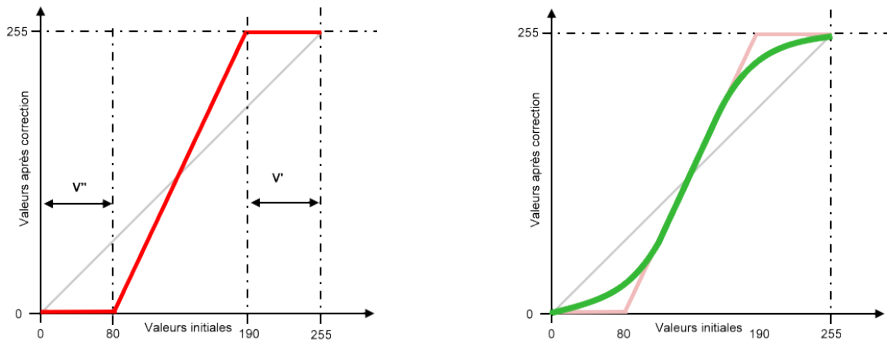
Deuxième conséquence : les points ayant une valeur comprise entre 0 et 100 deviendront noirs, puisque la valeur minimale est 0. Il y aura donc là aussi perte d'informations.

Exercice 4.2

Programmez en Python une fonction qui augmente la luminosité d'une image en additionnant une valeur fixe.

4.4.2 Augmenter le contraste

Pour rendre une image plus contrastée, il faut assombrir les points foncés et éclaircir les points clairs, par exemple comme dans les figures ci-dessous :



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

Exercice 4.3

Dessinez une courbe tonale qui va diminuer le contraste.

Exercice 4.4

Programmez en Python une fonction qui augmente le contraste d'une image, en se référant à la courbe rouge de la figure de gauche ci-dessus. Il s'agira en particulier de trouver l'équation de la droite entre les abscisses 80 et 190.

Programmez ensuite une autre fonction qui diminue le contraste (voir ex. 4.3).

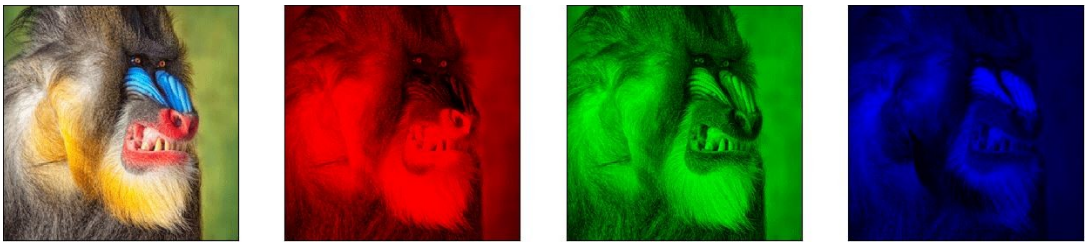
4.5. Quelques traitements classiques

Dans ce paragraphe, nous allons modifier les valeurs des composantes des pixels.

4.5.1. Rouge, vert et bleu

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 rouge.

De manière analogue, on peut obtenir des images vertes et bleues.



Exercice 4.5

Programmez en Python une fonction qui produit les trois images ci-dessus.

4.5.2. 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. En réalité, une image gif en niveaux de gris ne comptera pas 3 composantes mais une seule : un entier entre 0 (noir) et 255 (blanc).

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, la formule standard donnant le niveau de gris en fonction des trois composantes est :

```
gris = int(round(0.299*rouge + 0.587*vert + 0.114*bleu))
```

Exercice 4.6

Programmez en Python une fonction qui transforme une image en couleurs en une image en niveaux de gris grâce à la formule ci-dessus.

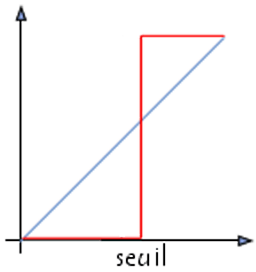
Comparez ensuite cette image avec celle obtenue en prenant comme formule :

```
gris = (rouge + vert + bleu) // 3
```

Observez en particulier les zones bleues de l'image en couleurs.

4.5.3. 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 **123**). Ainsi, si un pixel à une valeur supérieure au seuil (par exemple **150**), il prendra la valeur **255** (blanc), et si sa valeur est inférieure (par exemple **100**), il prendra la valeur **0** (noir).



Avec une image en couleurs, on fera de même avec les composantes rouge, verte et bleue. Il n'y aura alors que 8 couleurs possibles pour chaque pixel : blanc, noir, rouge, vert, bleu, magenta, jaune et cyan.

Exercice 4.7

Programmez en Python une fonction qui effectue le seuillage d'une image en couleurs avec un seuil donné en paramètre.

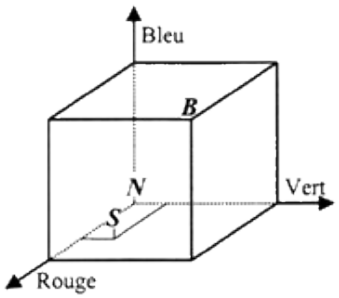
Comparez les résultats obtenus avec différents seuils.

4.5.4. Sépia

En photographie, le sépia est une qualité de tirage qui ressemble au noir et blanc, mais avec des variations de brun, et non de gris. La couleur sépia dans le système RVB est *S*(94, 38, 18).

Dans la transformation d'une image couleur en une image en nuances de sépia, on tient compte d'un seuil (0 < *seuil* < 255), qui sépare le sépia assombri et le sépia éclairci.

La transformation se fait alors pixel par pixel en deux temps. Pour chaque pixel, on calcule d'abord un niveau de gris qui est la moyenne *m* des intensités de rouge, vert et bleu. Puis, si le gris obtenu est foncé (*m* < *seuil*), on le remplace par une couleur du segment *NS*, couleur d'autant plus proche du noir que *m* est petit. Si le gris est plus clair (*m* > *seuil*), il est remplacé par une couleur du segment *SB*, couleur d'autant plus proche du blanc *B* que *m* est grand.



Exercice 4.8*

Programmez en Python une fonction qui transforme une image en couleurs en une image aux tons sépia.

4.5.5. Pixellisation

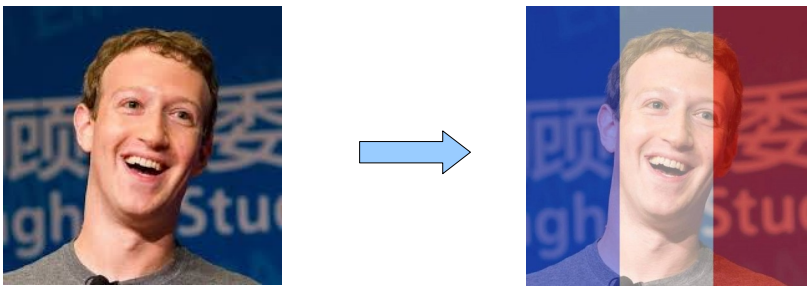
L'image est divisée en rectangles de la taille spécifiée (dans notre exemple 8×8 pixels, au centre, et 16×16 pixels, à droite). Chaque rectangle est ensuite rempli avec la couleur moyenne de la zone.

Exercice 4.9

Programmez en Python une fonction qui pixélise une image, la taille des gros pixels étant donnée en paramètre.

4.6. Les drapeaux

Quelques heures après les attentats de Paris du 13 novembre 2015, *Facebook* activait une option temporaire pour habiller les photos de profil de ses utilisateurs aux couleurs du drapeau français. Très vite, le réseau social s'est coloré de bleu, blanc, rouge pour marquer sa solidarité avec les victimes. Ce procédé pourrait fort heureusement être utilisé dans des circonstances moins dramatiques, par exemple pour marquer son soutien à son équipe sportive préférée.



Exercice 4.10

Écrivez des codes Python qui auront pour effet de superposer sur une image un des drapeaux ci-dessous (ou d'autres). Pour la Croatie, on se contentera de reproduire de damier rouge et blanc.

France Allemagne Suisse* Croatie*

4.7. Filtres

Un filtre est une transformation mathématique (appelée *produit de convolution*) permettant de modifier la valeur d'un pixel en fonction des valeurs des pixels avoisinants, affectées de coefficients. Les calculs sont faits pour chacune des trois composantes de couleur. Le filtre est représenté par un tableau (une matrice), caractérisé par ses dimensions et ses coefficients, dont le centre correspond au pixel concerné.

4.7.1. Lissage

Le lissage rend l'image plus floue. On dit que c'est un filtre passe-bas. Appliquer ce tableau revient en fait à remplacer la valeur de chaque pixel par la moyenne des 9 pixels formant un carré.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9


Rappelons que les valeurs des composantes des pixels sont des nombres entiers compris entre 0 et 255. Si les nouvelles valeurs ne sont plus des entiers, il faudra les arrondir.

Exercice 4.11

Programmez en Python une fonction qui lisse une image selon le procédé expliqué ci-dessus.

4.7.2. Accentuation

À l'inverse, le tableau ci-après rendra l'image plus nette. C'est un filtre passe-haut.

Attention ! Il peut arriver que la nouvelle valeur ne soit plus comprise entre 0 et 255. Il faudra donc toujours prendre $\min(x, 255)$ et $\max(x, 0)$, où x est la nouvelle valeur.

0	–0.5	0
–0.5	3	–0.5
0	–0.5	0


Exercice 4.12

Programmez en Python une fonction qui rend une image plus nette, selon le procédé expliqué ci-dessus.

4.7.3. Gradient (filtre de Sobel)

Pour faire simple, l'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords.

–1	0	1
–2	0	2
–1	0	1

Exercice 4.13

Reprenez le programme qui transforme une image en couleurs en une image en niveaux de gris (ex 4.7), puis appliquez le filtre de Sobel à cette image.

4.7.4. Le filtre médian

La technique de filtre médian est largement utilisée en traitement d'images numériques, car elle permet de réduire le bruit tout en conservant les contours de l'image.

L'idée principale du filtre médian est de remplacer chaque pixel par la valeur médiane de son voisinage.

Considérons neuf pixels en niveaux de gris, dont une valeur est aberrante (ici 255) :

2	4	12
2	255	3
7	9	3

Le filtre médian va d'abord trier ces valeurs par ordre croissant : 2, 2, 3, 3, **4**, 7, 9, 12, 255 et prendre la valeur médiane (la cinquième valeur), ici la valeur 4. La sortie du filtre donnera :

2	4	12
2	4	3
7	9	3

Exercice 4.14

La première ajoutera du bruit à une image : un certain nombre de pixels aléatoires seront remplacés par des pixels blancs.

La seconde permettra d'éliminer ce bruit en utilisant un filtre médian. Pour une image en couleurs, on appliquera ce filtre au trois composantes RVB séparément.

Image bruitée

Image filtrée

4.8. Transformations géométriques

Rappel
Dans nos images de taille l x h, le pixel de coordonnées (0 ; 0) se trouve dans le coin supérieur gauche tandis que le pixel du coin inférieur droit a pour coordonnées (l–1 ; h–1).

Nous donnerons ci-dessous les formules pour réaliser des transformations géométriques en notation matricielle. Ce sont les formules classique dans un système de coordonnées cartésien ; il faudra les adapter car, dans nos images, le système de coordonnées est différent.

4.8.1. Symétries axiales

Une symétrie d'axe horizontal (passant par le centre de l'image) est définie par la formule :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}$$

où $(x_c\ ;\ y_c)$ sont les coordonnées du centre de l'image, $(x\ ;\ y)$ les coordonnées d'un pixel dans l'image originale et $(x'\ ;\ y')$ les coordonnées de ce pixel dans l'image destination.

Autrement dit, $x' = x$, et $y' = -y + 2y_c$

Symétrie d'axe horizontal

Symétrie d'axe vertical

Exercice 4.15

Programmez en Python une fonction qui effectue une symétrie d'axe horizontal et une autre qui effectue une symétrie d'axe vertical.

4.8.2. Homothétie

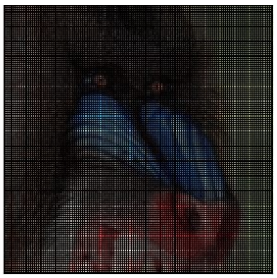
Une homothétie de centre $(x_c\ ;\ y_c)$ et de facteur λ est définie par la formule :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \cdot \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}$$

où $(x_c\ ;\ y_c)$ sont les coordonnées du centre de l'image, $(x\ ;\ y)$ les coordonnées d'un pixel dans l'image originale et $(x'\ ;\ y')$ les coordonnées de ce pixel dans l'image destination.

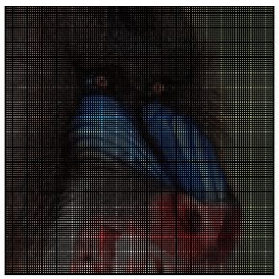
Exercice 4.16

Programmez en Python une fonction *zoom* qui effectue une homothétie. Les paramètres de cette fonction seront le centre et le facteur d'échelle.



$\lambda = 0.5$ $\lambda = 2$

On constate qu'avec un facteur d'échelle $\lambda > 1$, l'image résultat est **creuse** (il y a beaucoup de pixels noirs). Pour éviter cela il faut procéder autrement : pour faire un zoom, nous allons parcourir les pixels de l'image destination (qui seront tous noirs au départ), puis appliquer la transformation inverse pour retrouver le pixel antécédent.



Zoom *Smart zoom*

Exercice 4.17

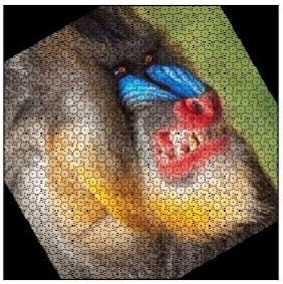
Programmez en Python une fonction *smart_zoom* qui effectue une homothétie en évitant le problème des images creuses. Les paramètres de cette fonction seront le centre et le facteur d'échelle.

4.8.3. Rotation

Une rotation d'angle α (dans le sens trigonométrique) centrée en $(x_c\ ;\ y_c)$ est définie par la formule :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}$$

où $(x_c\,;\,y_c)$ sont les coordonnées du centre de l'image, $(x\,;\,y)$ les coordonnées d'un pixel dans l'image originale et $(x'\,;\,y')$ les coordonnées de ce pixel dans l'image destination.


Rotation de 30° *Smart rotation*

Exercice 4.18

Programmez en Python une fonction *rotation* qui effectue une rotation d'angle α autour du centre de l'image (carrée).

Vous constaterez un problème similaire à celui des images creuses de l'homothétie. Programmez ensuite une fonction *smart_rotation* qui élimine ce problème.

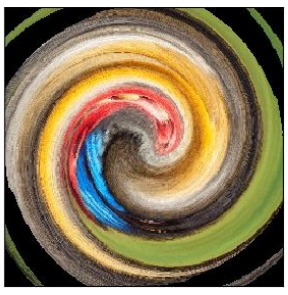
4.8.4. Twist

Le *twist* consiste en une rotation du plan de centre $(x_c\,;\,y_c)$ dont l'angle α est fonction de la distance du pixel au centre. Il est défini par la formule :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}$$

avec $\alpha = \rho\sqrt{(x-x_c)^2+(y-y_c)^2}$





$\rho = 0.01$ $\rho = 0.04$

Exercice 4.19

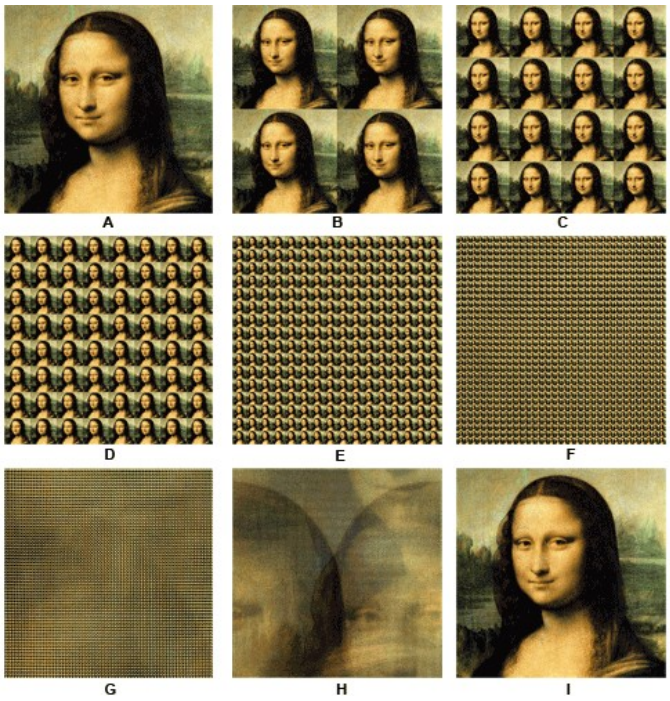
Programmez en Python une fonction *twist* qui effectue un twist de paramètre ρ autour du centre de l'image (carrée).

Vous constaterez un problème similaire à celui des images creuses de la rotation. Programmez une fonction *smart_twist* qui élimine ce problème.

À partir d'une image « twistée », peut-on retrouver l'originale ? Essayez !

4.8.5. Le photomaton

Le texte et les images de ce paragraphe proviennent de [1].

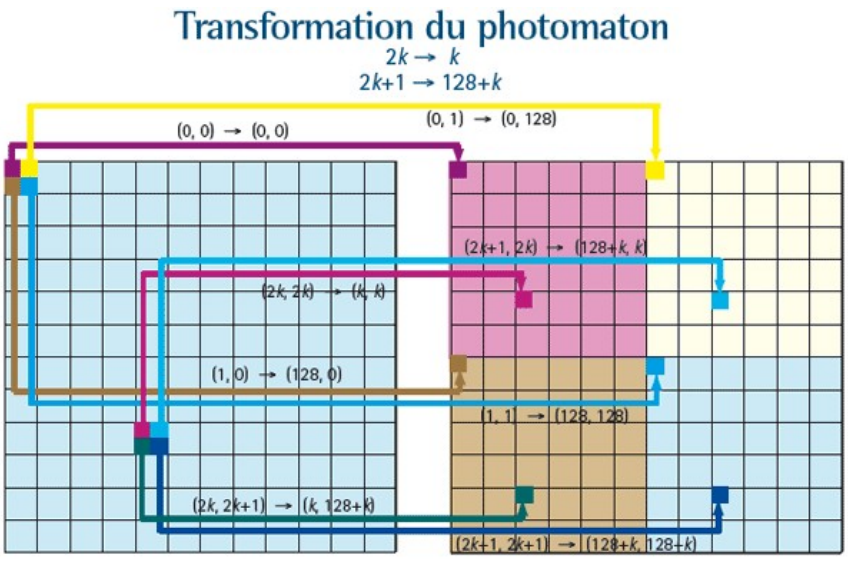


Regardez attentivement la série de 9 images A, B, C, D, E, F, G, H, I. Chacune a été obtenue à partir de la précédente en réduisant la taille de l'image de moitié, ce qui a donné quatre morceaux analogues qu'on a placés en carré pour obtenir une image ayant la même taille que l'image d'origine. Le nombre de pixels a été exactement conservé et en fait, on a seulement déplacé chacun des pixels (sans en changer la couleur). Précisément on a découpé l'image initiale en paquets carrés de quatre pixels (2x2), puis pour chaque paquet carré de quatre pixels, on a utilisé celui du haut à gauche pour l'image réduite de Mona Lisa en haut à gauche, celui en haut à droite pour l'image au haut à droite de Mona Lisa, etc. Cette opération produit bien quatre versions réduites de Mona Lisa. Cette transformation s'appelle la *transformation du photomaton*.

L'image B comporte 4 Mona Lisa. L'image C en comporte 16. L'image D en comporte 64, etc. Il se produit quelque chose d'étrange car, au bout de 9 étapes, l'image de Mona Lisa est réapparue. Précisons que c'est bien la même transformation qui a été utilisée pour déduire les unes après les autres les images de la série.

Exercice 4.20

L'image utilisée comporte 256 lignes et 256 colonnes numérotées de 0 à 255. La transformation du photomaton consiste à réaliser l'opération suivante :



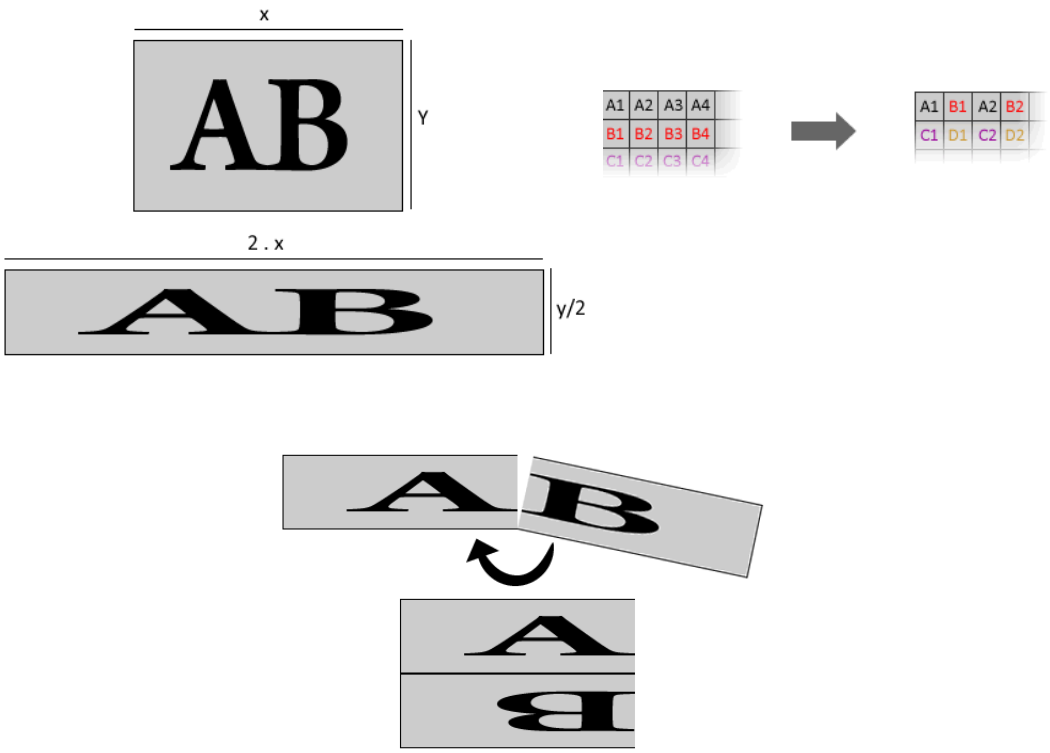
Le pixel (0, 0) reste donc en position (0, 0) ; le pixel (1, 0) passe en position (128, 0) ; le pixel en position (1, 1) passe en position (128, 128) ; le pixel en position (4, 5) passe en position (2, 130), etc. (pour un numéro pair $2k$ on passe à k, pour un numéro impair $2k+1$ on passe à $128+k$).

Programmez la transformation du photomaton. Faites en sorte que le programme fonctionne quelles que soient les dimensions de l'image (pourvu qu'il y ait un nombre pair de lignes et de colonnes).

Vérifiez qu'après 8 transformations du photomaton on retrouve bien l'image originale.

4.8.6. La transformation du boulanger

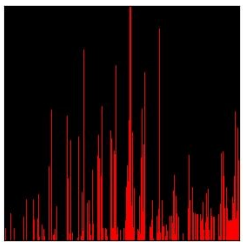
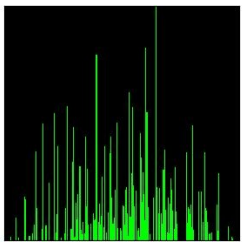
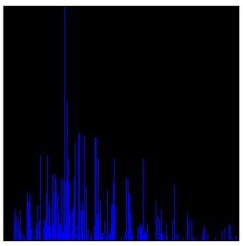
La transformation du boulanger est un mélange analogue au pétrissage par un boulanger qui étire une pâte jusqu'à ce qu'elle soit d'épaisseur moitié, puis la coupe en deux et superpose les deux moitiés pour lui redonner sa dimension initiale, et ainsi de suite.



Exercice 4.21*

Programmez la transformation du boulanger. Faites en sorte que le programme fonctionne quelles que soient les dimensions de l'image (pourvu qu'il y ait un nombre pair de lignes et de colonnes).

4.9. Histogrammes

L'histogramme d'une image mesure la distribution des valeurs 0 à 255.

Concrètement, l'histogramme d'une image à valeurs entières est construit de la manière suivante : pour chaque valeur d'une composante, on compte le nombre de pixels ayant la valeur.

En divisant chaque valeur de l'histogramme par le nombre total de pixels dans l'image on obtient un histogramme normalisé, qui correspond à une distribution de probabilité empirique (toutes les valeurs sont comprises entre 0 et 1 et la somme des valeurs vaut 1).

L'histogramme permet d'obtenir rapidement une information générale sur l'apparence de l'image. Une image visuellement plaisante aura généralement un histogramme équilibré (proche d'une fonction plate).

Notons enfin que l'histogramme ne contient aucune information spatiale et que des images très différentes peuvent avoir des histogrammes similaires.

Exercice 4.22

Écrivez un programme Python qui donne les histogrammes des trois composantes rouge, verte et bleue d'une image.

Dessinez l'histogramme dans la couleur de la composante correspondante, sur fond noir.

4.10. Combinaison d'images

Il est facile de combiner deux images de même taille. Il suffit pour cela de calculer une moyenne pondérée des pixels de même emplacement des deux images.

La formule ci-dessous illustre le calcul à réaliser :

$$\begin{aligned} r &= \alpha \cdot r_1 + (1-\alpha) \cdot r_2 \\ g &= \alpha \cdot g_1 + (1-\alpha) \cdot g_2 \\ b &= \alpha \cdot b_1 + (1-\alpha) \cdot b_2 \end{aligned} \qquad 0 \le \alpha \le 1$$


Exercice 4.23

Écrivez en Python une fonction permettant de mélanger deux images de mêmes dimensions selon la formule ci-dessus, avec α donné en paramètre.

Avec le même principe, il est possible de cacher numériquement une image « sous » une autre. Par exemple, on peut cacher le plan d'une base aérienne sous *Le moulin de la Galette* de Pierre-Auguste Renoir.

L'image 1 est l'image camouflante (la peinture de Renoir) et l'image 2 l'image à camoufler.
La formule ci-dessous montre comment calculer les pixels de l'image truquée :

$$r = r_1 + \alpha \cdot r_2$$
$$g = g_1 + \alpha \cdot g_2 \qquad 0 \le \alpha \le 1 \text{ (avec } \alpha \text{ petit, par exemple } \alpha = 1/64)$$
$$b = b_1 + \alpha \cdot b_2$$

Ce système de transmission de secret implique que les deux protagonistes possèdent l'image 1.

Pour faire réapparaître l'image dissimulée, il suffira de soustraire l'image du dessus :

$$r_2 = (r - r_1)/\alpha$$
$$g_2 = (g - g_1)/\alpha \qquad 0 \le \alpha \le 1$$
$$b_2 = (b - b_1)/\alpha$$

Remarque

Les composantes des pixels proches 255 peuvent poser problème. En effet, en leur additionnant un certain nombre, ils risquent de dépasser 255 et on devra les seuiller. Cela a pour conséquence que l'image 2 déchiffrée sera légèrement altérée par rapport à l'image 2 de départ.

Exercice 4.24

Écrivez un programme Python qui :
a) camoufle un QR-code sous une image de même taille,
b) retrouve le QR-code.

On peut aussi remplacer des pixels d'une certaine couleur par les pixels de l'autre image. C'est le principe du fond vert (ou bleu), communément utilisé au cinéma, à la télévision (pour la présentation de la météo par exemple) et par les youtubeurs.

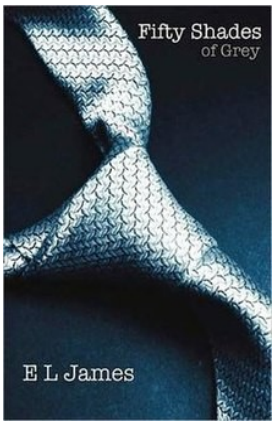
Exercice 4.25

Écrivez en Python une fonction permettant d'obtenir l'effet ci-dessous.


Sources

[1] Jean-Paul Delahaye, « Mona Lisa au photomaton » — *Images des Mathématiques*, CNRS, 2013, <https://images-archive.math.cnrs.fr/Mona-Lisa-au-photomaton.html>

[2] Étienne Coutant, Olivier Nocent, Frédéric Blanchard, « Algèbre linéaire et traitement des images », <https://fredbl.gitlab.io/algebre-lineaire-et-imagerie-numerique>




```
#050505  #0A0A0A  #0F0F0F  #141414
#191919  #1E1E1E  #232323  #282828
#2D2D2D  #323232  #373737  #3C3C3C
#414141  #494949  #4B4B4B  #505050
#555555  #5A5A5A  #5F5F5F  #646464
#669699  #6E6E6E  #737373  #787878
#7D7D7D  #828282  #878787  #8C8C8C
#919191  #999999  #9B9B9B  #A0A0A0
#A5A5A5  #AAAAAA  #AFAFAF  #B4B4B4
#B9B9B9  #BEBEBE  #C3C3C3  #C8C8C8
#CDCDCD  #D2D2D2  #D7D7D7  #DCDCDC
#E1E1E1  #E9E9E9  #EBEBEB  #F0F0F0
#F5F5F5  #FAFAFA
```