TP : Traitement d'images avec Numpy

Dans ce TP, on va manipuler des images, à l'aide des tableaux Numpy

1 Création d'un répertoire de travail

Avant de commencer ce TP, créez un répertoire spécifique dans votre espace personnel, par exemple `TP_image`.

2 Complément sur les tableaux Numpy

Exécutez le code suivant (manipulation de tableaux « standard » de type `list`):

```
T=[[0,1],[2,3]] # des tableaux dont les éléments sont des tableaux.
U=[[4,5],[6,7]]
print(T+U) # concaténation
print(2*T) # concaténation
```

Exécutez maintenant (manipulation de tableaux Numpy) :

```
import numpy as np #importé en haut de l'annexe
T2=np.array(T) # conversion en tableau numpy.
U2=np.array(U)
print(T2+U2) # addition terme à terme.
print(2*T2) # multiplication de tous les éléments par 2.
```

Remarquez qu'avec des tableaux numpy, le produit `T2*U2` ne produit pas d'erreur et a bien un sens ici : celui du produit terme à terme des tableaux, comme pour l'addition. Attention, pour avoir un sens, les opérations précédentes doivent se faire sur des tableaux ayant même taille :

```
T=np.array([[0,1],[2,3],[4,5]])
U=np.array([[4,5],[6,7]])
print(T+U) #addition d'un tableau 3*2 avec un tableau 2*2: erreur.
```

En numpy, les tableaux sont homogènes : ils représentent un tableau multi-dimensionnel d'éléments de même type. Pour accéder à, ou modifier un élément d'un tableau Numpy, cela fonctionne comme pour des listes de listes, essayez la séquence d'instructions :

```
T=np.array([[0,1],[2,3],[4,5]])
print(T[1])
print(T[1][0])
T[2][0]=10
print(T)
```

Pour copier un tableau, il faut utiliser la fonction `copy` de Numpy.

```
T=np.array([[0,1],[2,3],[4,5]])
T2=T # si un élément de T ou T2 est modifié alors cette modification sera aussi appliquée à l'autre
     # tableau: T et T2 sont deux références vers le même tableau en mémoire.
T3=np.copy(T) # si un élément de T ou T3 est modifié alors cette modification ne sera pas visible sur
              # l'autre tableau: les éléments ont été recopiés ailleurs en mémoire.
T[0][0]=6
print(T)
print(T2)
print(T3)
```

C'est à peu près tout ce qu'on a besoin de savoir !

3 Décomposition d'une image en pixels

Les images qu'on va utiliser sont des images dites matricielles. Elles sont composées d'une matrice (tableau) de points colorés appelés *pixels*. Le format qu'on utilisera est PNG en « couleurs vraies ». Chaque pixel est un triplet de nombres entre 0 et 255 : un nombre pour chaque couleur primaire rouge, vert, bleu. Un tel nombre est représentable sur 8 bits et s'appelle un *octet*. Il y a donc $256^3 = 2^{24} = 16777216$ couleurs possibles. On utilise ici la synthèse additive des couleurs : le triplet $(0, 0, 0)$ correspond à un pixel noir alors qu'un pixel blanc est donné par $(255, 255, 255)$. Un pixel « pur rouge » est codé par $(255, 0, 0)$.

Voici ci-dessous la décomposition d'une image quelconque [1] en ses trois composantes rouge, verte et bleue.

Remarquez que comme l'image est assez mauve dans l'ensemble (mélange de rouge et bleu) le contraste entre les parties claires et foncées est plus saisissant dans la teinte verte.

La séquence Python suivante indique comment récupérer un tableau numpy donnant un tableau tridimensionnel, de taille $h \times \ell \times 3$ où h est la hauteur de l'image et ℓ est sa largeur.

Récupération d'une liste de pixels sous forme de tableau 3D

```
from PIL import Image #importation du sous-module Image du module PIL
im=Image.open("\chemin\vers\image.png") #ouverture d'une image au format png dans Python.
tab=np.array(im)
```

Récupérez l'image `lena.png`. Enregistrez-là dans le répertoire que vous avez créé à cet effet. Comme c'est maintenant le répertoire de travail, vous pouvez normalement ouvrir l'image avec `Image.open("lena.png")`. Ouvrez l'image `lena.png` et chargez-la (le code est fourni). Exécutez également le code qui suit :

```
print(tab) #contrairement aux tableaux usuels, tout n'est pas affiché (heureusement !)
print(im.size) #im.size renvoie la taille de l'image (largeur x hauteur)
print(len(tab)) #nb de sous-tableaux de tab, c'est-à-dire nombre de lignes sur l'image
print(len(tab[0])) #nb de sous-sous-tableaux de tab, c'est-à-dire nombre de colonnes
print(len(tab[0][0])) #nb de couleurs additives utilisées, ici 3 : R, V et B
print(tab.shape) #renvoie un tuple contenant les éléments précédents (h, l, 3)
```

Dans la suite, on va jouer avec les images en les modifiant, Pour cela, il suffit de modifier le tableau numpy associé (ou d'en créer un nouveau). Voyons comment obtenir une image à partir d'un tableau numpy :

Obtenir une image à partir d'un tableau

```
nouvelle_image=Image.fromarray(tab)
nouvelle_image.show() # pour afficher l'image
nouvelle_image.save("nom_de_la_nouvelle_image.png") # pour l'enregistrer au format voulu
```

1. Pas vraiment quelconque... Cette photo issue d'un numéro de Playboy de 1972 est devenue l'image la plus utilisée pour les tests d'algorithmes de traitement d'images.

Le tableau Numpy `tab` doit être composé d'éléments `uint8` (c'est-à-dire d'entiers non signés[2] codés sur 8 bits, soit entre 0 et 255). Si ce n'est pas le cas (flottants, entiers codées sur 32 bits...), un message d'erreur comme `TypeError: Cannot handle this data type` s'affiche. Pour créer de nouvelles images, on procédera de deux façons :

— en modifiant un tableau Numpy obtenu à partir d'une image : le tableau a déja le bon type ;
— en créant un nouveau tableau pour le remplir. Dans ce cas, il faudra veiller à ce que le tableau ait le bon type :

* on peut créer un tableau de type `uint8` à partir d'un tableau `tab` qui n'est pas de ce type avec `np.uint8(tab)`.
* on peut créer un tableau rempli de zéros, de type `uint8` avec `np.zeros((h, l, 3), dtype="uint8")`. Par défaut à l'utilisation de `np.zeros`, le tableau créé est rempli de zéros flottants. Ici, on précise explicitement que l'on veut que les données soit de type `uint8`.

4 À vous de jouer

Dans ce TP, par convention on écrit des fonctions qui prennent en entrée des images et qui retournent des images. Par exemple la fonction suivante crée une image de la même taille que l'image passée en entrée, mais remplie de pixels noirs (elle est fournie) :

```
from PIL import Image #ces modules sont déja importés en haut de l'annexe
import numpy as np

def image_noire(im):
    t=np.array(im)
    h,l,r=t.shape #on sait que r=3
    for i in range(h):
        for j in range(l):
            for k in range(3):
                t[i][j][k]=0
    return Image.fromarray(t)
```

Remarquez que `0*t` aurait également créé un tableau de même taille rempli de zéros, mais on fera souvent usage de telles boucles imbriquées dans ce TP : i parcourt les indices des lignes, j parcourt les indices des colonnes, k parcourt $\{0,1,2\}$ qui sont les indices des pixels. Il faut que les boucles soient imbriquées pour faire parcourir à (i,j,k) toutes les valeurs $[\![0,h-1]\!]\times[\![0,\ell-1]\!]\times[\![0,2]\!]$. Une utilisation peut-être :

```
lena=Image.open("lena.png")
lena_noire=image_noire(lena)
lena_noire.show() #affichage
```

4.1 Décomposition d'une image en ses composantes

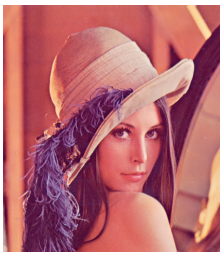
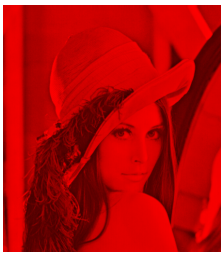
Question 1. Ecrire une fonction `composante_rouge(im)` retournant la composante rouge de l'image passée en paramètre. Servez-vous en pour calculer la composante rouge de `lena.png`. En travaillant sur le tableau associé à l'image `im`, il s'agit simplement de mettre à 0 les composantes 1 et 2 des pixels.

```
def composante_rouge(im):
    ...
    A COMPLETER
    ...

lena=Image.open("lena.png")
composante_rouge(lena).show()
```

Le résultat attendu est donné ci-dessous.

2. Entier non signé signifie entier naturel : la suite de 8 bits s'interprète comme vu en cours. Entier signé serait entier relatif en complément à deux.

 ⇝ 

Faire de même avec les composantes vertes et bleues (on écrira les fonctions associées).

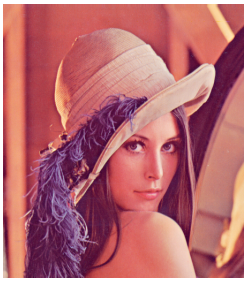
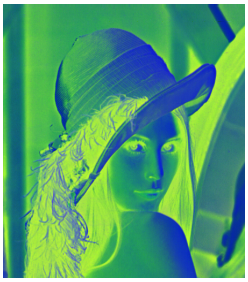
4.2 Négatif d'une image

Une image négative est une image dont les couleurs ont été inversées par rapport à l'originale ; par exemple le rouge devient cyan, le vert devient magenta, le bleu devient jaune et inversement. Les régions sombres deviennent claires, le noir devient blanc. Pour cela il suffit d'inverser les niveaux de chacune des couleurs primaires : un pixel initialement à (120, 10, 250) deviendra (135, 245, 5).

Question 2. Écrire une fonction `negatif(im)` retournant l'image négative de l'image passée en paramètre.

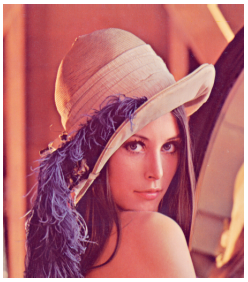
```
def negatif(im):
    ...
    A COMPLETER
    ...
```

Le résultat attendu pour Léna est le suivant :

 ⇝ 

4.3 Mise en place d'un cadre autour d'une image

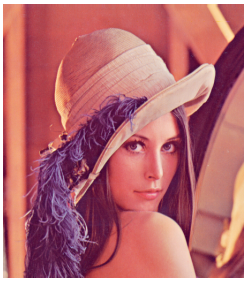
Question 3. Écrire une fonction `cadre_noir(im,ep)` prenant en entrée une image, ainsi qu'un (petit) entier `ep`, renvoyant une nouvelle image, identique à la précédente mais dont les pixels situés sur les bords de l'image ont été remplacés par des pixels noirs, sur une épaisseur `ep`.
Par exemple, le résultat attendu pour Léna et une épaisseur de 5 est donné ci-dessous.

 ⇝ 

Une fois votre fonction écrite, appliquez-la à Léna avec une épaisseur de 5.

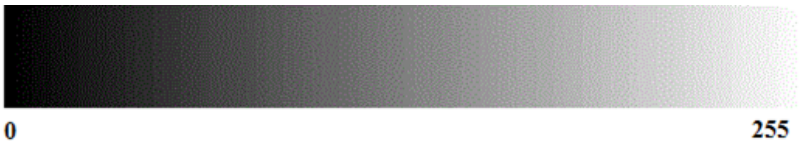
4.4 Rajout d'un cadre autour de l'image

Question 4. Écrire une fonction `rajout_cadre(im,ep)` fonctionnant comme précédemment, mais retournant une image de taille $(h + 2\text{ep}, \ell + 2\text{ep})$ (si l'image initiale a pour taille (h, ℓ)), le cadre ayant été ajouté à l'extérieur plutôt que par modification des pixels du bord. Voici le résultat avec une épaisseur de 20 pixels. Le plus simple est de créer un nouveau tableau à la bonne taille (avec `np.zeros`, dont le comportement a été expliqué plus haut, attention à avoir le type `uint8`), et de le remplir. Rappel : le pixel numéroté $(0, 0)$ correspond au coin en haut à gauche d'une image, et (i, j) au pixel situé à l'intersection de la i-ème ligne et de la j-ème colonne.


$\rightsquigarrow$


4.5 Conversion d'une image couleur en image en niveau de gris

Dans une image en niveaux de gris, les trois composantes R, V, B de chaque pixel ont la même valeur. Celle-ci vaut 0 pour un pixel noir, 255 pour un pixel blanc..

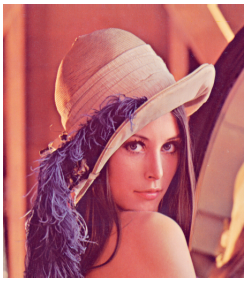


L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

$$\text{gris} = \lfloor 0.299 \cdot \text{rouge} + 0.587 \cdot \text{vert} + 0.114 \cdot \text{bleu} \rfloor$$

où $\lfloor . \rfloor$ désigne la partie entière.

Question 5. Ecrire une fonction `niveau_gris(im)` renvoyant une nouvelle image, en niveau de gris. Les trois niveaux R, V, B d'un pixel sont égaux au niveau de gris donné par la formule ci-dessus.
Le résultat attendu pour Léna est donné ci-dessous.


$\rightsquigarrow$


Utilisez votre fonction pour obtenir une image que vous sauvegarderez sous le nom `lena_gris.png`

4.6 Image au format « B »

Dans une image en niveau de gris, les niveaux de rouge, vert et bleu étant identiques, les informations sont redondantes. Il existe un format pour stocker des images en noir et blanc (inutile de répéter trois fois la même

valeur !). Il est possible de sauvegarder la matrice des pixels non pas en associant à chaque pixel un triplet de niveau (R,V,B) mais une valeur unique égale au niveau de gris. Le « B » du nom de format signifie simplement « black ».

On pourra créer une matrice numpy à 2 dimensions (et non plus 3), représentative d'une image noire à l'aide de l'instruction :

```
tabng=np.zeros((h, l), dtype="uint8") # h est la hauteur de l'image, l la largeur
```

On placera ensuite à chaque emplacement du tableau `tabng[i][j]` le niveau de gris calculé du pixel de la ligne `i` et colonne `j`.

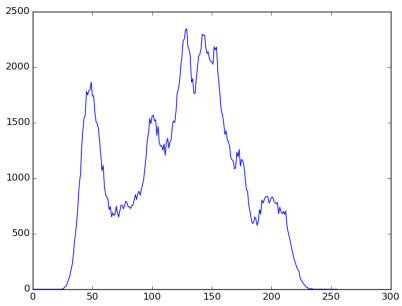
Question 6. Ecrire une fonction `niveau_gris2(im)` faisant la même chose que la fonction `niveau_gris(im)`, mais renvoyant une image au format « B » et non plus « RGB ». L'insruction `Image.fromarray` renvoie une image au bon format à partir d'un tableau de dimension 2.

Utilisez votre fonction pour obtenir une image que vous sauvegarderez sous le nom `lena_gris2.png` (le résultat attendu est visuellement identique au précédent). Cliquez droit sur chacune des deux images en niveau de gris créées et comparez leur taille en ko. Notez que la deuxième image est moins volumineuse que la première !

4.7 Histogramme d'une image

L'histogramme d'une photo permet de compter le nombre de pixel d'un niveau de gris donné. Celui de la photo `lena_gris2.png` est donné ci-dessous.

⇝



Question 7. Écrire une fonction `histo(im)`, qui prend en argument une image en niveau de gris au format « B » (décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris). Cette fonction doit renvoyer une liste de taille 256 : en première position (indice 0), le nombre de pixels noirs (gris 0), en deuxième position (indice 1), le nombre de pixels gris 1, ..., en dernière position (255), le nombre de pixels blancs (gris 255).

Appliquez votre code à l'image `lena_gris2.png`, préalablement chargée. Vous utiliserez ensuite le module `matplotlib` pour tracer l'histogramme. Petit rappel pour tracer un graphe :

```
import matplotlib.pyplot as plt
plt.plot(X,Y) #X et Y sont les listes (ou tableaux numpy) d'abscisses et d'ordonnées des points à tracer
plt.show() #pour afficher
```

Une image qui a un histogramme tout à gauche, c'est-à-dire une image très sombre, est image dite « bouchée ». Une image qui a un histogramme tout à droite, c'est-à-dire une image avec des lumières très vives, est image dite « brûlée ».

4.8 Modifier la luminosité d'une image

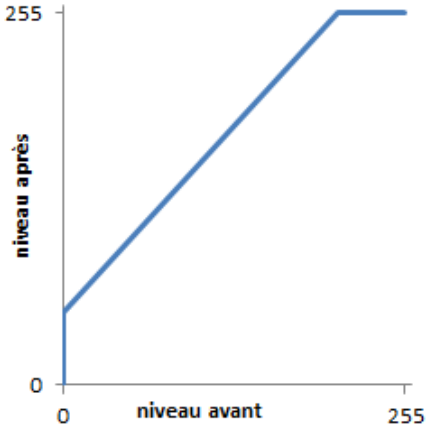
Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe à tous les niveaux. Pour diminuer la luminosité il faudra au contraire soustraire une valeur fixe à tous les niveaux.

Question 8. Écrire une fonction `change_luminosite(im,d)`, qui prend en argument une image en niveau de gris décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris et un entier entre 0 et 255, valeur du décalage du niveau de gris. Cette fonction renvoie une nouvelle image. Attention : on prendra garde que si l'on essaie de mettre un entier dans un tableau dont les éléments sont de type `uint8`, celui-ci est pris modulo 256. On convient que pour une valeur de pixel p, si $p + d > 255$ il faut stocker 255, et si $p + d < 0$, il faut stocker 0.

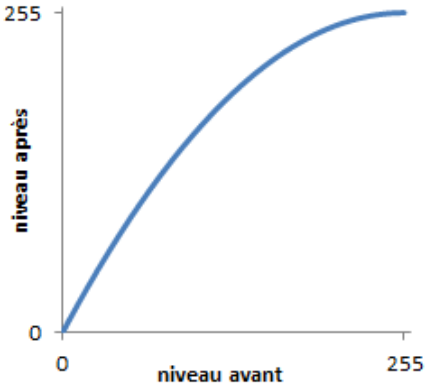
Le résultat attendu pour `lena_gris2.png` avec un décalage de 50 est donné ci-dessous.


$\rightsquigarrow$


Remarque : il est très mauvais d'augmenter ainsi la luminosité : avec un décalage de 50, il n'existera plus aucun point entre 0 et 50, et les points ayant une valeur supérieure à 205 deviendront des points parfaitement blancs, puisque la valeur maximale possible est 255. La nouvelle image contient des zones brûlées. Plutôt que d'utiliser la fonction $p \mapsto \begin{cases} p+50 & \text{si } p < 205. \\ 255 & \text{sinon.} \end{cases}$, il vaut mieux utiliser une fonction « presque bijective » de forte croissance au voisinage de 0 et de très faible croissance au voisinage de 255, comme sur le graphe ci-dessous :



Transformation initiale



Une meilleure transformation

4.9 Augmentation du contraste par dilatation de l'histogramme

Une méthode relativement simple pour augmenter les contrastes est de s'arranger pour que l'histogramme de la nouvelle image occupe toute la plage de valeurs $[\![0, 255]\!]$. Pour ce faire, on peut, en considérant l'histogramme comme une fonction $H : [\![0, 255]\!] \to \mathbb{N}$:

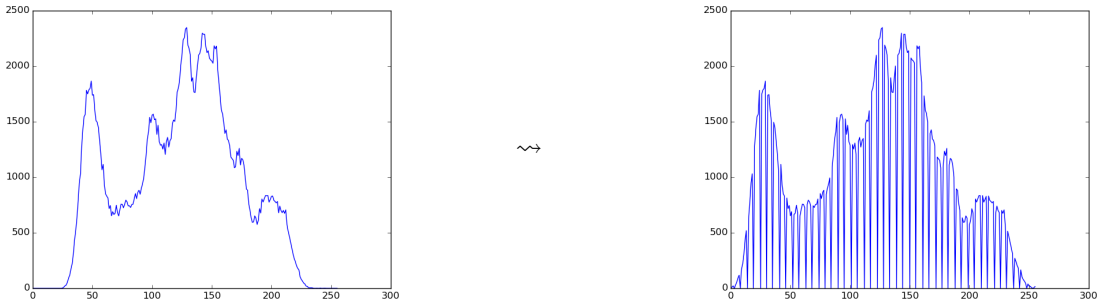
— repérer dans l'histogramme de l'image initiale les indices $i_{\min}$ et $i_{\max}$ tels que pour $i < i_{\min}$ ou $i > i_{\max}$, on ait $H(i) < s$, avec s un petit entier seuil (par exemple $s = 3$). Autrement dit, il y a strictement moins de s pixels de couleur gris i pour tous les i strictement inférieurs à $i_{\min}$ ou strictement supérieurs à $i_{\max}$;

— appliquer la transformation affine $x \mapsto \frac{256 \times (x - i_{\min})}{i_{\max} - i_{\min}}$ à chaque pixel gris x de l'image, en arrondissant à l'entier le plus proche et en remplaçant les valeurs négatives par 0 et les valeurs supérieures à 256 par 255.

En appliquant cette transformation à l'image `lena_gris2.png`, avec seuil $s = 3$, on obtient le résultat suivant :


$\rightsquigarrow$


Ceci est très visible sur l'histogramme, voici comment celui-ci a été transformé :



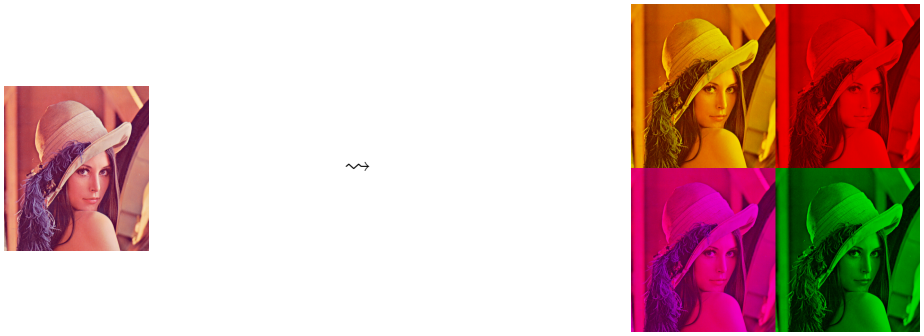
Question 9. Écrire une fonction `augmente_contraste(im,s)` prenant en entrée une image au format « B » (donnée par niveaux de gris) et le seuil s, et renvoyant l'image obtenue par le procédé décrit précédemment. Testez votre fonction avec l'image `lena_gris2.png`.

Remarque : en fait, on peut augmenter également les contrastes sur une image au format RVB, en calculant d'abord l'image en niveau de gris associée, puis son histogramme, et en appliquant ensuite la transformation décrite ci-dessus à l'image au format RVB. Voici le résultat sur l'image de Léna originelle :

4.10 Effet popart (Andy WARHOL)

Question 10. Écrire une fonction `popart(im)` permettant de réaliser un effet popart sur une image. Pour cela appliquer un filtre de couleur à l'image (ce filtre peut être un filtre rouge, vert, bleu ou de tout autre couleur mélange de ces trois couleurs primaires).

Le résultat attendu est donné ci-dessous (en haut à gauche, image avec un filtre jaune (R,V,B)=(255,255,0), en haut à droite image avec un filtre rouge (R,V,B)=(255,0,0), en bas à gauche image avec un filtre magenta (R,V,B)=(255,0,255) et en bas à droite image avec un filtre vert (R,V,B)=(0,255,0)).



Indices (à ne lire que si vous êtes bloqué !) :

1. Ecrire une fonction `filtre(t,p)` ayant deux arguments : le premier est une matrice de pixel (un tableau), et le deuxième un tableau de trois élément correspond au niveau R, V, B de la couleur qui doit dominer sur l'image résultante. ;
2. Créer 4 tableaux `tab_hg` (pour tableau en haut à gauche), `tab_hd`, `tab_bg` et `tab_bd` correspondant au tableau initial auxquels les filtres de couleur ont été appliquées ;
3. Créer un tableau numpy à partir des 4 tableaux précédents (on pourra utilement lancer `help(np.concatenate)`...) ;
4. Convertir en image.