1ère NSI PROJET : traitement d'images

Objectifs :

- Appliquer les bases de programmation Python
- Appliquer plusieurs algorithmes pour transformer des images numériques
- Utiliser une bibliothèque et sa documentation

1/ Introduction et installation :

Une image est constituée de points appelés pixels. Chaque pixel a une couleur représentée par un triplet (r, v, b) correspondant aux composantes rouge, verte et bleue. Chaque composante est un nombre codé sur 1 octet (entre 0 et 255 donc).
Pour manipuler les images numériques, vous allez utiliser la bibliothèque Python de traitement d'images PIL.

Pour cela,ouvrir un terminal et entrez la commande :
pip3 install PILLOW (ou pip install PILLOW)

Vous pourrez maintenant, en utilisant les images données dans le dossier partagé, exécuter le fichier test_PIL.py qui vous aidera à utiliser la bibliothèque PIL.
Vous pouvez utiliser les images disponibles dans le dossier partagé ou d'autres images de votre choix.

2/ Votre mission

Dans un nouveau fichier que vous nommerez projet_images.py, vous écrirez les fonctions demandées ci-dessous :

- niveaux_de_gris(image)
- miroir(image)
- flou(image)
- fonctions de stéganographie : cache_image(im1, im2) et retrouve_image(image)
- et une fonction de votre choix.

Une fois les fonctions écrites, vous écrirez une fonction main() dans laquelle vous proposerez à l'utilisateur l'une ou l'autre des 4 transformations et afficherez dans la même image l'image initiale et l'image transformée.
Pour la stéganographie, vous afficherez l'image initiale, l'image à cacher, l'image contenant l'image cachée puis l'image secrète restaurée.

<u>Description des fonctions</u> :

- la fonction niveaux_de_gris prend en argument une image couleur et renvoie l'image en niveaux de gris ; on affecte à chaque pixel la moyenne des 3 composantes rouge, verte et bleue.

- la fonction miroir prend en argument une image et renvoie l'image à laquelle on a appliqué l'effet miroir, comme illustré ci-dessous

- la fonction flou prend en argument une image et renvoie l'image floutée obtenue en réalisant pour chaque pixel la moyenne de la valeur du pixel et de celles des 8 pixels voisins.

<u>Description de la stéganographie</u>

La stéganographie consiste à dissimuler une information dans une autre.

Ici, on utilise le fait que l'œil ne distingue pas les faibles différences de couleur pour cacher une image dans une autre. Pour simplifier, voyons ce qui se passe sur une seule composante de couleur, disons le rouge (on appliquera ensuite le même traitement aux trois couleurs).

Prenons 2 images de même dimension. Pour chacune des coordonnées (x,y), on va mélanger une partie du pixel (x,y) de l'image 1 avec le pixel (x,y) de l'image 2. L'image obtenue sera très proche de l'image 1, mais l'image 2 sera « cachée » à l'intérieur.

Imaginons qu'on veut mélanger un pixel dans l'image 1 dont l'octet rouge vaut **190** c'est-à-dire 10111110

avec l'octet de l'image 2 qui vaut 121, c'est-à-dire 01111001

L'idée est de recombiner les 5 chiffres les plus significatifs du premier pixel avec les 3 chiffres les plus significatifs de l'image 2. Les chiffres significatifs sont les plus importants, ceux qui se trouvent à gauche. De nos 2 octets d'origine
10111 110 et 011 11001

on ne garde que les chiffres les plus significatifs, en mettant ceux de la première image en tête. Ce mélange des deux octets d'origine produit l'octet suivant:
10111 011

On remarque qu'on a peu modifié la valeur par rapport à celle de l'image 1, parce qu'on a gardé les 5 bits les plus significatifs. Sur notre exemple, la différence est seulement de 3: la nouvelle valeur est 187, ce qui donne un rouge **comme ce texte** au lieu de 190 dans l'image originale, ce qui donne un rouge **comme celui-ci** qui est très proche à l'œil nu. Au pire, on transforme les 3 derniers bits de **000** à **111**, ou vice-versa, ce qui fait une différence de 7, au plus. En résumé, la fusion des 2 images ressemble beaucoup à la première image.

Qu'a-t-on gagné ? Ce qu'on a gagné est qu'à partir de l'octet qu'on a créé dans l'image transformée:
10111 011

1. on n'a pas beaucoup modifié la 1ère image
2. on peut retrouver une couleur proche de celle du pixel **de la 2ème image** grâce aux 3 derniers bits, 011. On sait donc que la valeur du pixel original de la 2ème image se situe entre 011 00000 et 011 11111 .

L'erreur est au maximum de 32. Pour reconstruire approximativement le pixel original, on peut compléter la valeur de façon intermédiaire, en ajoutant 16. À partir d'un pixel de l'image modifiée, on reconstruit donc l'approximation du pixel de l'image cachée:
01110000

Ce nombre vaut **112** au lieu de **121** dans l'image 2 qu'on voulait cacher. On a donc fait une erreur de 9 qui reste acceptable. Au pire, on fera une erreur de 16. Cela reste suffisant pour retrouver l'image cachée, même si la perte de précision la dégradera.