Rudiments de traitement d'images avec Python

# Qu'est-ce qu'une image

Quand on y réfléchit, définir une image n'est pas si facile ! Je dirais, de façon imparfaite, que c'est un ensemble de points qui représentent une forme au sens large du terme (paysage, personne, objet, dessin, etc.). La chose importante est que ce n'est pas un objet continu au sens mathématique du terme mais discret. Les points d'une image sont généralement appelés des pixels (du terme anglais "picture element") mais rigoureusement ce terme devrait être réservé aux images numérisées.

Une image peut être représentée sous la forme d'une matrice NxM, où chaque élément serait un pixel. N et M définissent la taille de la matrice et donc la taille de l'image numérisée encodée par la matrice. Son traitement va se résumer pour l'essentiel à du calcul matriciel et de la manipulation d'éléments d'une matrice.

Un pixel est un nombre, qui code son aspect dans l'image, sa couleur et parfois sa transparence. La nature informatique du nombre, short, integer, ou long, dépend donc du nombre de couleurs que l'on veut coder. Par exemple, si l'on veut coder une image en noir et blanc, un seul bit est suffisant pour exprimer un pixel. Si l'on désire coder une image en 256 couleurs, il faudra 8 bits et pour 64K couleurs il faudra 16 bits. Ainsi, une image de 64K couleurs sera codée par une matrice de pixels, chacun étant codé par un nombre de 16 bits (en Python un entier sera donc largement suffisant..).

Pour résumer, dans une image, un pixel est caractérisé par sa position qui est indiquée par les indices ligne et colonne de la matrice et par sa valeur, qui code la couleur du pixel.

## Image et fichier image

Une erreur extrêmement répandue jusque dans les cours de traitement d'images est de confondre l'image et le fichier qui contient, entre autre, l'image. On lit souvent des phrases du type « une image jpeg » ou « une image png » : ce sont des abus de langage ! Ce n'est pas l'image, qui est je le rappelle une matrice de nombres, qui est jpeg ou png, c'est le fichier qui contient l'image !

Quelle est la différence ? Un fichier « image » contient plus qu'une image. Il contient dans son entête des données, plus exactement des métadonnées, qui décrivent la forme, la dimension, les traitements de compressions éventuellement subis par l'image, son auteur, la date de création et beaucoup d'autres données. D'ailleurs, si vous êtes curieux, vous pouvez calculer la taille d'une image donnée en octets (assez facile) puis la comparer à la taille en octets du fichier qui stocke l'image : c'est instructif !

Lorsqu'on parle d'une image au format jpeg, on désigne en fait un fichier avec une extension .jpeg ou .jpg, qui contient une image ayant subi une compression avec un algorithme jpeg. Les caractéristiques de la compression sont indiquées dans les métadonnées du fichier, mais pas dans la matrice qui code l'image ! C'est la même chose pour les autres formats de fichiers images, que je vais lister ci-dessous.

## Les principaux formats de fichiers images

Ils sont très nombreux, aussi, je ne vais citer que les formats les plus courants, ceux que vous voyez tous les jours :

* jepg (Joint Photographic Expert Group) : C'est un format de compression d'images avec perte de données, sans doute le plus répandu. Les images compressées peuvent être noir & blanc ou couleur, avec 8, 16 ou 32 bits par pixel. Son grand avantage est de produire des images de faible poids (volume de fichier petit).
* png (Portable Network Graphic) : C'est un format de compression d'images, sans perte de données, moins efficace que jpeg. Les couleurs sont codées sur 24 bits, donc 16,7 millions de couleurs. Il est très utilisé sur le net car il permet de la compression en gardant une très bonne qualité d'image.
* gif (Graphics Interchange Format) : format d'image sans compression, mais avec seulement 8 bits par pixel (codage de 256 couleurs). Les images sont de taille faible. On le rencontre beaucoup sur le net.
* tiff (Tagged Image File Format) : format d'image sans compression, mais avec 16 ou 32 bits par pixel, donc des images très volumineuses. Il n'est pas utilisé sur le net mais pour des usages professionnels comme les imprimeurs ou les publicitaires.
* bmp (BitMap): le format ancestral, sans compression de données, avec des images énormes. A oublier, sauf besoins locaux précis.

En traitement d'images, il se peut que vous ayez besoin d'informations sur l'image stockée dans un fichier image. Vous les trouverez dans l'en-tête du fichier. Certaines librairies d'imagerie permettent d'extraire ces en-têtes et de les décomposer. Vous pouvez aussi chercher la description des en-têtes de fichier et lire vous-même ces informations.

# Les outils Python pour le traitement d'images

## Notre image de test

Pour nos manipulations, nous travaillerons sur une image couleur au format RGB, c'est à dire que chaque pixel est codé par un 3-tuple (R,G,B) qui indique le poids, compris entre 0 et 255, de chaque canal de couleur : Red, Green et Blue. Il aurait été beaucoup plus simple de travailler sur une image en niveau de gris avec des pixels codés sur un seul entier, mais cela aurait été moins drôle !

Voici l'image en question :

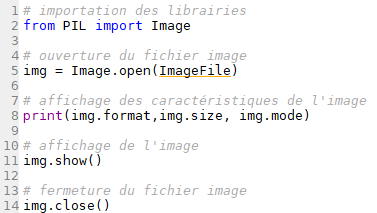


Vous aurez reconnu un Hawkeye E-2C de la flottille 4F, un AWACS embarqué sur porte-avion. Les connaisseurs auront reconnu un « chien jaune » du R91 « Charles de Gaulle ».

## La librairie Python PIL

La librairie PIL (Python Imaging Library) fournit les outils nécessaires pour les manipulations d'images que nous aborderons dans ces pages. Ces manipulations sont simples et il existe des libraries plus complètes pour aborder les fonctions avancées de traitement des images. Vous trouverez une description des fonctions disponibles dans PIL (1.1.7) sur ce site et bien d'autres...

À titre d'introduction , voyons le script Python TIProg1.py qui ouvre et affiche une image et ses principales caractéristiques (taille, compression, mode vidéo):



Voilà, c'est simple et de bon goût ! Je vais utiliser cette trame de programme dans tous les scripts qui suivent.

# Quelques opérations sur les images avec Python

## Manipulations simples sur une image

### Inversion d'une image

Techniquement, cette transformation est simple : il suffit de calculer le complément à 255 (au blanc) pour chaque composante R,G et B d'un pixel. Il faut donc parcourir tous les pixels de l'image et leur appliquer cette transformation.

Le script TIProgNégatif.py se présente comme suit, dans sa partie traitement, sachant que la partie ouverture du fichier image est inchangée. Je commence par récupérer le nombre de lignes et de colonnes de l'image. Une précision : le pixel en haut à gauche est d'indice (0,0) et celui en bas à droite (colonne-1,ligne-1).

colonne,ligne = img.size

Je crée une image de mêmes caractéristiques que l'image source pour stocker l'image issue du traitement :

imgF = Image.new(img.mode,img.size)

Puis vient la boucle de traitement, en fait deux boucles imbriquées qui parcourent l'image d'abord selon les lignes, puis selon les colonnes. Ce sens est conventionnel. Pour chaque pixel, je lis son 3-tuple avec la méthode getpixel(). Je calcule la valeur du nouveau pixel en appliquant la transformation, dans ce cas le complément à 255 pour chaque composante du pixel. Enfin, je crée le pixel transformé dans l'image finale imgF avec la méthode putpixel(). Ce qui nous donne le code :

for i in range(ligne):

for j in range(colonne):

pixel = img.getpixel((j,i)) # récupération du pixel

# on calcule le complement à MAX pour chaque composante - effet négatif

p = (255 - pixel[0], 255 - pixel[1], 255 - pixel[2]) # ( Rouge, Vert, Bleu )

# composition de la nouvelle image

imgF.putpixel((j,i), p)

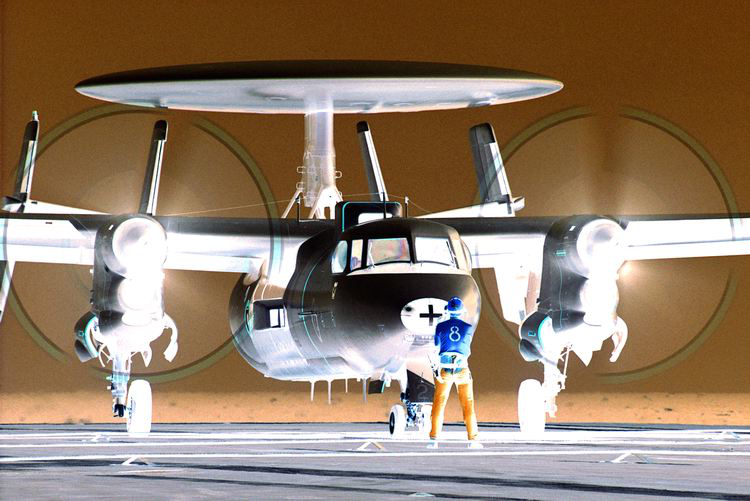
Le script s'achève par l'affichage de l'image en négatif puis la fermeture de l'image originale par les instructions :

imgF.show()

img.close()

Retenez bien ce schéma de code, car nous le retrouverons dans tous les scripts qui suivent. Seul l'algorithme de la transformation appliquée aux pixels sera différent.

Voilà les résultats obtenus par le script TIProgNégatif.py :



### Symétrie d'une image

Les opérations de transformation géométrique sont relativement simples, ne consistant qu'en un déplacement de pixels, sans modification. Ici, nous voulons obtenir un effet miroir, c'est à dire une symétrie par rapport à un axe vertical, le bord droit de l'image en l'occurrence.

Écrire ce programme. Solution dans TIProgSymétrie.py



### Transformer une image couleur en niveaux de gris

Intuitivement, on « sent » que pour obtenir le niveau de gris d'un pixel couleur, il faudrait mélanger les composantes du 3-tuple (R,G,B). De vieux souvenirs de cours de colorimétrie en optique. Soit, mais dans quelles proportions ? À parts égales ? Si vous faites l'expérience, vous verrez qu'on obtient bien une image en niveaux de gris, mais avec un je ne sais quoi de pas convaincant !

En cherchant, j'ai trouvé que les proportions recherchées étaient normalisées : c'est la CIE (Commission Internationale de l'Éclairage) qui normalise ce genre de chose, histoire de savoir de quoi l'on parle. Et dans sa norme 709, elle dit que pour les images naturelles les poids respectifs doivent être 0.2125 \* R + 0.7154 \* G + 0.0721 \* B. Nous allons donc utiliser cette répartition dans notre code.

Autre chose, qui semble évident : dans un pixel en niveaux de gris, les composantes du 3-tuples ont la même valeur, celle calculée ci-dessus.

Écrire ce programme. Solution dans TIProgNiveauGris.py



## Filtrage

Le filtrage est une opération fondamentale en traitement du signal et donc en traitement des images. Il existe deux types de filtrage d'un signal : le filtrage spatial et le filtrage fréquentiel.

On peut assimiler un filtre à un opérateur qui réaliserait une opération sur un pixel en fonction de la valeur de ce pixel et des pixels de son voisinage. En ce sens, les transformations que nous avons abordées plus haut sont des filtrages, par exemple le passage au négatif ou le passage en niveau de gris. L'opérateur peut être linéaire ou non linéaire. On obtiendra donc des filtres linéaires ou non linéaires en fonction de la nature de l'opérateur.

### Le filtrage des couleurs

C'est un exemple simple de filtrage, couramment pratiqué en photographie : on ne garde qu'une seule couleur dans une image. Il peut s'agir par exemple de ne conserver que le canal rouge, vert ou bleu dans une image. Cette opération est généralement destinée à faire ressortir des détails dans une image.

Écrire un programme qui se contente de mettre à zéro les composantes R et B du pixel. Solution dans TIProgFiltre1.py.



Transformer le programme pour avoir la composante bleue seule, puis pour retirer la composante rouge seulement.

### Les filtres par convolution

En filtrage d'images, le plus classique est d'appliquer sur un pixel un opérateur qui dépend du voisinage de ce pixel. La valeur du pixel sera modifiée en fonction de la valeur de ses voisins immédiats. Il s'agit de filtrage par convolution, car il utilise un outil du calcul matriciel qui s'appelle le produit de convolution.

Cette technique de filtrage consiste à considérer notre image comme une matrice de pixels que l'on va convoluer avec une autre matrice, plus petite, qui est le masque de convolution. C'est cette matrice qui va décider de la nature du filtre. En traitement du signal, on dit que cette matrice est la réponse impulsionnelle du filtre. Ici, nous allons utiliser une matrice 3x3, dont on fera varier la valeur des éléments pour obtenir différents types de filtre.

Analysons le script TIProgFiltreConvolution.py qui est un exemple de filtre par convolution. Tout d'abord, commençons par définir le masque de convolution de notre filtre. Techniquement, cela consiste à initialiser une matrice que j'appellerai Filtre :

Filtre = [[-1,-2,-1],[-2,16,-2],[-1,-2,-1]]

Les coefficients du masque correspondent à un filtre de contraste, mais nous en verrons d'autres plus loin. En fait, définir un filtre, c'est modifier les coefficients de cette matrice, le reste du programme reste inchangé, ce qui est plutôt pratique.

Le cœur du programme est la fonction Convolution2D dont voici le code :

def Convolution2D(Filtre,TPix,x,y):

p0 = p1 = p2 = 0

for i in range(-1,1):

for j in range(-1,1):

p0 += Filtre[i+1][j+1]\*TPix[y+i,x+j][0]

p1 += Filtre[i+1][j+1]\*TPix[y+i,x+j][1]

p2 += Filtre[i+1][j+1]\*TPix[y+i,x+j][2]

# normalisation des composantes

p0 = int(p0/9.0)

p1 = int(p1/9.0)

p2 = int(p2/9.0)

# retourne le pixel convolué

return (p0,p1,p2)

Nous passons à la fonction Convolution2D plusieurs paramètres : le masque de convolution (la matrice Filtre) ; la matrice image contenue dans TPix et les coordonnées (x,y) du pixel à convoluer.

La fonction contient essentiellement l'implémentation de l'algorithme de convolution. Mathématiquement, pour un filtre de dimensions 3x3, la convolution du pixel TPix(x,y) s'écrit:



Pour chaque canal de couleur, il s'agit de calculer la composante du pixel TPix(x,y) en fonction des composantes de ses voisins pondérées par les coefficients du filtre. Puis vient la normalisation où l'on divise par 9, le nombre d'éléments de la matrice, afin que la valeur de chaque composante reste comprise entre 0 et 255. Enfin, je créé le pixel convolué avec les trois composantes calculées et la fonction le retourne au programme appelant.

La boucle de traitement reste très simple :

TabPixel = img.load()

for x in range(1,ligne-1):

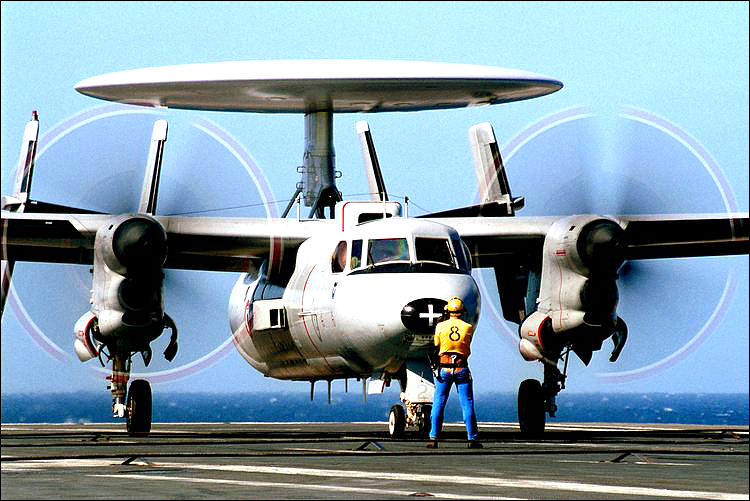
for y in range(1,colonne-1):

p = Convolution2D(Filtre,TabPixel,x,y)

imgF.putpixel((y,x),p)

Je charge l'image dans la matrice TabPixel en utilisant la méthode load(). Puis je parcours la matrice pixel par pixel. Pour chaque pixel, je procède à sa convolution en appelant la fonction Convolution2D puis je stocke le pixel convolué dans imgF, l'image filtrée.

Voici le résultat obtenu avec le filtre de convolution défini ci-dessus :



### Un filtre passe-haut

Un filtre passe-haut est un filtre qui accentue les fréquences hautes par rapport aux fréquences basses. Dans une image, cela se traduit par l'accentuation des détails.

Pour obtenir un filtre passe-haut, nous utiliserons la matrice Filtre :

Filtre = [[0,-4,0],[-4,18,-4],[0,-4,-0]]

Voici le résultat obtenu avec ce filtre :



Je vous invite à faire varier les valeurs des coefficients de la matrice de convolution et de vérifier l'effet sur l'image. Par exemple, quel est l'effet d'une modification de l'élément Filtre[2][2] ?

### Un filtre passe-bas

Un filtre passe-bas est un filtre qui accentue les fréquences basses par rapport aux fréquences hautes. Dans une image, cela se traduit par l'adoucissement des détails et la réduction du bruit.

Pour obtenir un filtre passe-bas, nous utiliserons la matrice Filtre :

Filtre = [[1,1,1],[1,6,1],[1,1,1]]

Voici le résultat obtenu avec ce filtre :



Là aussi, faites varier les composantes du filtre, expérimentez !

## Détection de contours

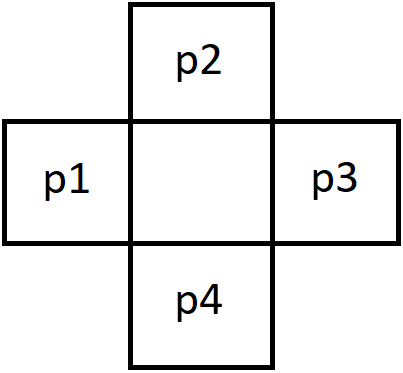
La reconnaissance de formes dans une image est une composante importante de l'analyse d'images. Elle se décompose en plusieurs étapes qui consistent à extraire les contours des objets dans l'image afin de les reconnaitre ou d'en détecter le mouvement. La première de ces étapes est la mise en évidence des contours des objets dans l'image. C'est cette étape que nous allons aborder très succinctement.

Un contour définit la limite d'un objet dans une image. Cette limite est caractérisée par un changement dans l'image : un changement de couleur ou de contraste. Ce changement se traduit dans la valeur des pixels qui sont localisés de part et d'autre de la limite. Nous sommes donc à la recherche d'un moyen de détecter et de localiser un changement. Les mathématiques nous donnent ce moyen sous la forme de la différentiation. On utilise habituellement des outils comme le gradient et le laplacien, bien connus des élèves de prépa, pour détecter ce changement. Mais ici, nous allons faire un peu plus simple en gardant le même principe.

Considérons un pixel p(i,j) dans une image couleur. Ce pixel est-il semblable, de même couleur, que ses voisins ? Si non, quelle est la différence de couleur entre lui et ses voisins ? Est-elle grande, ce qui signifierait qu'il est situé à la limite d'un objet ? Que signifie une « grande » ou une « petite » différence ? Comment la mesurer pratiquement ? C'est ce que nous allons essayer de traduire en algorithme.

Ce problème a fait l'objet de très nombreuses recherches. Il existe des algorithmes très efficaces et compliqués pour résoudre ce problème, surtout s'agissant d'images en couleurs. Mais il existe aussi des moyens simples, pas très performants mais utiles pour comprendre. Voyons la solution rudimentaire que je vous propose et ses résultats.

Le principe est de récupérer la valeur de chaque pixel avoisinant pour chaque pixel de l'image.



Puis de mesurer la différence, la « distance », entre notre pixel de référence et ses voisins en utilisant une fonction de norme standard, que vous reconnaissez sûrement ! Il existe bien d'autres normes, mais je vais au plus simple :

def Norme(p1,p2,p3,p4):

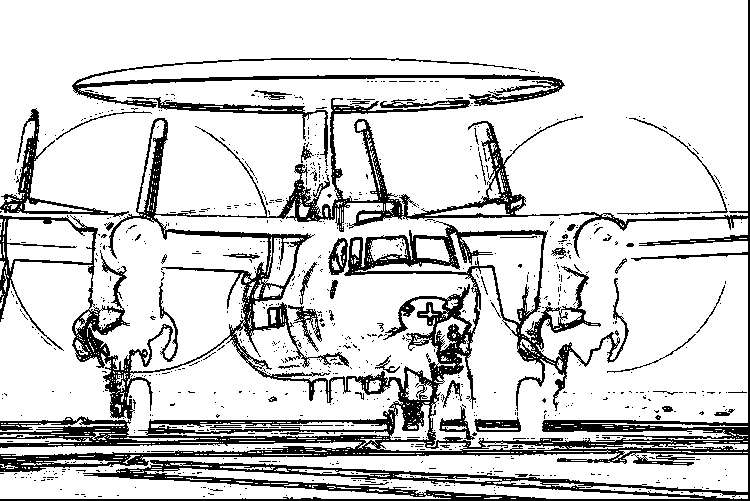
n = sqrt((p1-p3)\*\*2 + (p2-p4)\*\*2)

return n

Et c'est là que nous avons un petit problème ! Je passe à mon fonction Norme() les 4 pixels dont je veux évaluer la distance. Ce serait très simple si les pixels étaient codés sur un entier, comme dans une image en gris. Mais dans une image couleur, chaque pixel est un 3-tuple ! Il faudrait donc coder une norme avec trois variables par pixel. C'est faisable, mais assez compliqué. Donc, il va falloir ruser ! Il suffit de transformer notre image de référence en image à niveaux de gris avec l'algorithme vu précédemment. Et vous savez maintenant que les composantes d'un 3-tuple d'une image en niveaux de gris sont identiques !

Après avoir calculé la distance entre mon pixel courant et ses voisins, je décide si ce pixel est sur un contour ou pas à l'aide d'un seuillage. S'il est inférieur au seuil, c'est à dire pas très « distant » de ses voisins, je décide qu'il n'est pas élément d'un contour et je trace le en blanc, sinon, je le trace en noir, comme un contour.

Voilà pour le principe. Écrire un programme qui applique ce principe avec un seuil de 30. Solution dans TIExtractionContours.py.



Une dernière chose à propos de la détection de contours : on peut très bien la faire avec un filtrage convolutif, vu plus haut, en utilisant les filtres convenables: un filtre laplacien, de Sobel, Prewitt, Freeman, Kirsch et autres. Google vous renseignera sur le sujet.

# Tâche finale : Faites une œuvre d’art à la « Andy Warhol ».