TP1 : Initiation au traitement d'images
Afficher et créer une image avec OpenCV

Introduction

OpenCV (Open Source Computer Vision Library) est l'une des bibliothèques les plus utilisées pour les applications de Computer Vision. OpenCV-Python est l'API python pour OpenCV.
OpenCV-Python est non seulement rapide car l'arrière-plan est constitué de code écrit en C/C++, mais il est également facile à coder et à déployer (en raison du wrapper Python au premier plan). Cela en fait un excellent choix pour exécuter des programmes de Computer Vision intensifs en calcul. Ces algorithmes peuvent réaliser beaucoup de tâches telles que la détection et la reconnaissance de visages, l'identification des objets et bien d'autres. Vous pouvez aussi l'utiliser pour manipuler une image dans le but d'extraire des informations.

Ce premier TP est une initiation au traitement d'images en utilisant la bibliothèque OpenCV.
Notre objectif pour l'instant n'est pas d'en connaître toutes les fonctions, mais d'apprendre à lire, afficher, créer, modifier et sauvegarder une image.

1. Manipuler une image avec OpenCV

Commençons par importer les bibliothèques avec lesquelles nous allons travailler. Importez OpenCV **cv2**, **matplotlib.pyplot** (pour afficher les images) et la bibliothèque **numpy**.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

1.1. Importer une image

Chargeons l'image à partir d'un fichier avec la fonction imread() de OpenCV (cv2). Nous appellerons cv2.imread() et nous lui passerons le chemin vers le fichier image Cat.jpg :

```
image = cv2.imread('Cat.jpg')
# informations
print( 'classe :', type(image) )
print( 'type :', image.dtype )
print( 'taille :', image.shape )
print(image[0,0])                # accède à la valeur du premier pixel
```

Quel résultat obtenez-vous ?
Remarquez que la fonction imread() génère un vecteur de classe *numpy.ndarray* contenant des valeurs de type uint8 (entier) de 3 dimensions.

1.2. Afficher l'image

Essayons maintenant d'afficher l'image avec matplotlib (vous pouvez afficher l'image avec la commande cv2.imshow(), mais dans ce tutoriel nous préférons utiliser la bibliothèque matplotlib pour afficher les images) : Avec imshow(), faites attention à ne pas faire crasher votre kernel, utilisez cette syntaxe:

```
cv2.imshow("image de chat", image)  #fenêtre visualisant l'image
cv2.waitKey(0)                 #attend l'appui sur une touche du clavier
cv2.destroyAllWindows() #efface toutes les fenêtres ouvertes par imshow
```

Avec imshow de matplotlib:

```
plt.imshow(image)
plt.show()
```

Le résultat est affiché dans votre Notebook même après exécution. (more convenient for us)
Remarquez que les couleurs de cette image sont complétement différentes de l'image originale (et du résultat de imshow), **pourquoi** ?

La commande plt.imshow() affiche une image en mode RGB (rouge, vert, bleu), par contre la fonction **imread()** renvoie une image en mode **BGR (bleu, vert, rouge).** Il faut donc réorganiser les canaux de l'image pour pouvoir l'afficher correctement. Nous utilisons la fonction split() de OpenCV qui renvoie un vecteur numpy de 2 dimensions pour chaque canal et la fonction merge() pour les réassembler dans le bon ordre.
OpenCV imread, imwrite et imshow fonctionnent avec l'ordre BGR, il n'est donc pas nécessaire de changer l'ordre lorsque vous lisez une image avec cv2.imread et que vous voulez ensuite l'afficher avec cv2.imshow.
Alors que BGR est utilisé de manière cohérente dans OpenCV, la plupart des autres bibliothèques de traitement d'images utilisent l'ordre RGB. Si vous voulez utiliser imshow de matplotlib mais lire l'image avec OpenCV, vous devrez convertir de BGR en RGB.

```
#How to convert BGR to RGB: solution1
plt.imshow(image[..., ::-1]) # affiche la matrice de triplets RGB
plt.show()
#How to convert BGR to RGB: solution2
b, g, r = cv2.split(image)
image2 = cv2.merge([r, g, b])
plt.imshow(image2)
plt.show()
#How to convert BGR to RGB: solution3 more frequently used
img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.show()
```

1.3. Sauvegarder l'image

Finalement, vous pouvez enregistrer le vecteur numpy sous forme de fichier image avec la méthode imwrite() de cv2. Le format de l'image sauvegardé est automatiquement déterminé à partir de l'extension :

```
cv2.imwrite('Cat_bis.jpg', image)
```

CONCLUSION: En manipulant les images avec la bibliothèque opencv, faites attention lors de la lecture et de l'ecriture ou des transformations qui manipulent les canaux ! (BGR/RGB)

1.4. Image en nuance de gris/niveaux de gris

Pour convertir une image couleur en une image en nuance de gris, utilisez la fonction cvtColor() de cv2 qui prend l'image originale et l'attribut COLOR_RGB2GRAY en paramètres. Le script est le suivant :

```
grey_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
plt.imshow(grey_image, cmap = 'gray') #gray: colormap de nuances de gris
plt.show()
print("taille de l'image en nuance de gris :", grey_image.shape)
```

Désormais, vous avez un seul canal (plus de RGB). Votre matrice est de taille NxM, et non NxMx3

1.5. Changer la taille d'une image (sous-échantillonnage)

```
# Read the image using imread function
image = cv2.imread('Cat.jpg')
print(image.shape)              # affiche les dimensions de l'image

# let's downscale the image using new  width and height
down_width = 150
down_height = 100
down_points = (down_width, down_height)
resized_down = cv2.resize(image, down_points, interpolation=
cv2.INTER_LINEAR)
# Display images
plt.imshow(resized_down[..., ::-1])
plt.show()
```

1.6. Crop d'une image

```
# Cropping an image
cropped_image = image[25:75, 150:300]  #spécifier l'intervalle des indices

# Display cropped image
plt.imshow(cropped_image[..., ::-1])
plt.show()
# Save the cropped image
cv2.imwrite("Cropped Image.jpg", cropped_image)
```

1.7. Appliquer des rotations et des translations

```
# Reading the image
image = cv2.imread('cables.jpg')
height, width = image.shape[:2]
```

Rotation :

```
#compute the rotation point, which in this example, will be the center of
the image.
```

```
# Dividing height and width by 2 to get the center of the image
center = (width/2, height/2) # center of rotation axis
# use cv2.getRotationMatrix2D() to get the rotation matrix
rotate_matrix = cv2.getRotationMatrix2D(center=center, angle=45, scale=1)
# Rotate the image using cv2.warpAffine
rotated_image = cv2.warpAffine(src=image, M=rotate_matrix, dsize=(width,
height))
```

```
# visualize the original and the rotated image
plt.imshow(image[..., ::-1])
plt.show()
plt.imshow(rotated_image[..., ::-1])
plt.show()
```

Translation :

```
# get tx and ty values for translation, you can specify any value of your
choice
tx, ty = width / 4, height / 4

# create the translation matrix using tx and ty, it is a NumPy array
translation_matrix = np.array([ [1, 0, tx],[0, 1, ty]], dtype=np.float32)
# apply the translation to the image
translated_image = cv2.warpAffine(src=image, M=translation_matrix,
dsize=(width, height))
# display the original and the Translated images
plt.imshow(image[..., ::-1])
plt.show()
plt.imshow(translated_image[..., ::-1])
plt.show()
```

2. Créer votre propre image avec numpy et OpenCV

```
from PIL import Image
# Creating the 144 X 144 NumPy Array with random values
#img = np.random.randint(255, size=(144, 144), dtype=np.uint8)
img = np.zeros([5,5,3], dtype=np.uint8)
img[:,:,0] = np.ones([5,5])*64
img[:,:,1] = np.ones([5,5])*128
img[:,:,2] = np.ones([5,5])*192
# Converting the numpy array into image
img = Image.fromarray(img)
plt.imshow(img)
plt.show()
# Saving the image
img.save("Image_from_array.png")
```