NAS-DIP-pytorch

basé sur ce dépôt git:
https://github.com/YunChunChen/NAS-DIP-pytorch

Et sur le papier suivant:
https://arxiv.org/pdf/2008.11713v1

Installation:

Le projet datant de 2020, il est nécessaire de mettre à jour le code. J'ai aussi eu à modifier le code de sorte à ce qu'il soit plus agréable à utiliser.

J'ai tout d'abord mis à jour les importations de scikit: comme le code date d'il y a approximativement 5 ans, un imports est devenu obsolète et nécessitait d'être changé.

```
from skimage.metrics import peak_signal_noise_ratio as compare_psnr
```

au lieu de:

```
from skimage.measure import compare_psnr
```

J'ai aussi eu à ajouter du code pour faciliter le chargement des fichiers dans le code: Les images ont plusieurs canaux dépendant de leurs modes (en RGB elles en ont 4, en RGBA 2 et en L une), or le code ne prenait en compte que les images avec 1 channel dans les réseaux de neurones. J'ai donc dû adapter:

```
mode = img_pil.mode
        if mode == 'RGB':
            channels = 3
        elif mode == 'RGBA':
            channels = 4
```

```
        elif mode == 'L':
            channels = 1
        else:
            channels = None
```

```
if args.net == 'default':
            from models.skip import skip
            net = skip(num_input_channels=args.input_depth,
                       num_output_channels=channels,
```

En l'état, le code générait automatiquement des masques de bernouilli:

```
            img_mask    = get_bernoulli_mask(img_pil, 0.50)
            img_mask_np = pil_to_np(img_mask)

        img_masked  = img_np * img_mask_np
        mask_var    = np_to_torch(img_mask_np).type(dtype)
```

Ce qui n'était pas utile car nous voulions utiliser nos propres masques. J'ai donc ajouté du code pour prendre cela en charge:

```
mask_img = Image.open('data/inpainting/yello_mask_GT.png').convert('L')
        if mask_img:
            img_mask_np = np.array(mask_img)
            print("Dimensions du masque (mask_np) :", img_mask_np.shape)
            img_mask_np =
nn.ReflectionPad2d(1)(np_to_torch(img_mask_np))[0].numpy()
            #img_mask_np = img_mask_np.astype(np.float32) / 255.0
        else:
            img_mask    = get_bernoulli_mask(img_pil, 0.50)
            img_mask_np = pil_to_np(img_mask)

        img_masked  = img_np * img_mask_np
        mask_var    = np_to_torch(img_mask_np).type(dtype)
```

Après avoir effectué ces changements, j’ai débuté ma session de test.

Test #1

à cet instant, je n’avais pas encore compris le fonctionnement du modèle. J’ai donc juste tenté de l’utiliser avec les masques générés automatiquement et j’ai observé les rendus.

 ---> 

Ce résultat s’explique par le fait que l’image, lorsqu’elle fût chargée, n’avait qu’un seul canal.

Test #2
J’ai ensuite commencé à utiliser des masques fait à la main



Comme on peut le voir ici, le masque inversé (utilisé dans le projet ResShift) ne donne aucune différence. J'ai donc utilisé les masques utilisés de base.

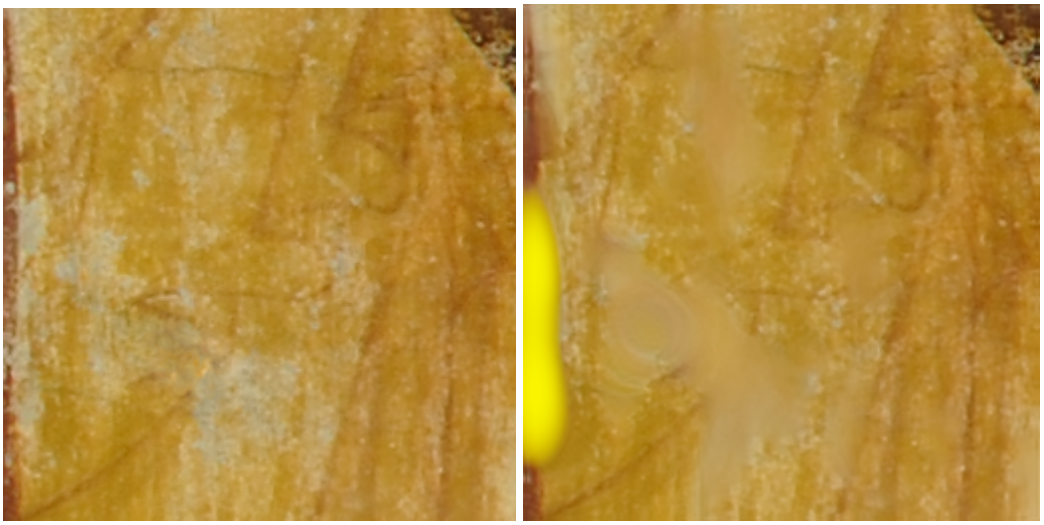
Test #3

C'est à ce point que je me suis rendu compte que ce code nous offrait plusieurs options de réseaux de neurones. Jusque là, j'utilisais le réseau par défaut (un réseau en U). Il y avait aussi l'option d'utiliser un NAS (expliqué dans le papier) et un Multicast.

J'ai aussi compris les différents arguments utilisables dans ce réseau.

Test #4: i_NAS = 10

Test #5: i_NAS = 5



Test #6: i_NAS = 100

Test #7: i_NAS = 200 num_iter=20000

Ce résultat est le meilleur que j’ai obtenu.

voici ci-dessous le tableau des arguments conseillés pour faire tourner le script du code:

Paramètre	Valeur conseillée	Pourquoi c'est important
`--optimizer`	`adam`	Stable et efficace sans dataset
`--num_iter`	`15000` à `20000`	Permet reconstruction fine sur fresques complexes
`--lr`	`0.0005` à `0.001`	Petit pas pour pas rater les détails
`--noise_method`	`noise`	Fonctionne bien pour tous les cas
`--input_depth`	`32` ou `64`	Plus = plus de détails, mais + lourd
`--batch_size`	`1`	Obligatoire (image par image)
`--random_seed`	`42`	Pour garder des résultats reproductibles
`--net`	`NAS`	Meilleure qualité globale pour restaurer les fresques
`--i_NAS`	`10`, `50`, `100`	Archis NAS bien testées, éviter les bizarres (249–251)
`--reg_noise_std`	`0.01` ou `0.03`	Moins si le masque est propre, plus si la fresque est bruitée
`--save_png`	`1`	Sauvegarde automatique des résultats visuels