

BIG DATA & DATA SCIENCE

USE CASE 4 : Deep Dream

Co-financé par :

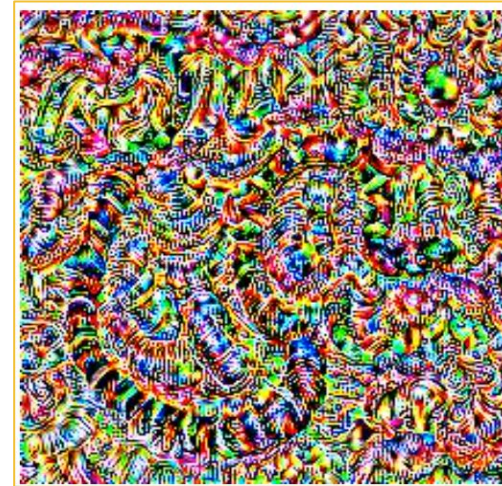
Use cases réalisés par les masters :

Sommaire

- Présentation du use case
- Préparation de l'environnement
- Importation des librairies
- La fonction `__init__`
- Les couches du réseau
- La fonction `dream`
- La fonction `__main__`
- Visualisation des résultats

1. Présentation du use case

Ce use case porte sur l'analyse d'images afin de créer de nouvelles images à partir d'une image de référence comme sur l'exemple ci-dessous :






L'entièreté du code source open source sur lequel nous allons travailler et des traitement d'images supplémentaires sont disponible sur le github ci-dessous :



<https://github.com/utkuozbulak/pytorch-cnn-visualizations>

Pour la suite du use case, nous allons utiliser GoogleColab : <https://colab.research.google.com>

2. Préparation de l'environnement

- ▶  generated
- ▶  input_images
- ▶  sample_data
-  misc_functions.py

- ▼  input_images
 -  snake.png

- ▼  generated
 - ▶  ddream_l

Dans l'environnement de GoogleColab, créer deux dossiers « generated » et « input_images ».

Télécharger snake.png sur le lien suivant et mettre l'image dans input_images :

https://github.com/utkuozbulak/pytorch-cnn-visualizations/blob/master/input_images/snake.png?raw=true

Créer un fichier misc_functions.py et y importer ou y copier coller le code disponible sur :

Créer un dossier « ddream_l » dans le dossier « generated ».

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

3. Importation des librairies

Dans le nouveau notebook, on importe les librairies nécessaires au projet :

```
import os
from PIL import Image
import torch
from torch.optim import SGD
from torchvision import models

from misc_functions import preprocess_image, recreate_image, save_image
```

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

4. La fonction `__init__`

Dans une classe `DeepDream`, nous créons tout d'abord le constructeur `__init__` qui sera la méthode appelée lors de la construction de l'objet :

```
class DeepDream():
    """
    Produces an image that minimizes the loss of a convolution
    operation for a specific layer and filter
    """
    def __init__(self, model, selected_layer, selected_filter, im_path):
        self.model = model
        self.model.eval()
        self.selected_layer = selected_layer
        self.selected_filter = selected_filter
        self.conv_output = 0
        # Generate a random image
        self.created_image = Image.open(im_path).convert('RGB')
        # Hook the layers to get result of the convolution
        self.hook_layer()
        # Create the folder to export images if not exists
        if not os.path.exists('../generated'):
            os.makedirs('../generated')
```

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

5. Les couches du réseau

On crée la fonction hook layer qui va permettre de sélectionner les couches du réseau de neurones :

```
def hook_layer(self):  
    def hook_function(module, grad_in, grad_out):  
        # Gets the conv output of the selected filter (from selected layer)  
        self.conv_output = grad_out[0, self.selected_filter]  
  
    # Hook the selected layer  
    self.model[self.selected_layer].register_forward_hook(hook_function)
```

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

6. La fonction dream

On crée la fonction dream qui va traiter l'image :

```
def dream(self):  
    # Process image and return variable  
    self.processed_image = preprocess_image(self.created_image, True)  
    # Define optimizer for the image  
    # Earlier layers need higher learning rates to visualize whereas later layers need less  
    optimizer = SGD([self.processed_image], lr=12, weight_decay=1e-4)  
    for i in range(1, 251):  
        optimizer.zero_grad()  
        # Assign create image to a variable to move forward in the model  
        x = self.processed_image  
        for index, layer in enumerate(self.model):  
            # Forward  
            x = layer(x)  
            # Only need to forward until we the selected layer is reached  
            if index == self.selected_layer:  
                break
```

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

6. La fonction dream

On y minimise la fonction de perte (fonction qui évalue l'écart entre les prédictions réalisées par le réseau de neurones et les valeurs réelles des observations utilisées pendant l'apprentissage).

```
# Loss function is the mean of the output of the selected layer/filter
# We try to minimize the mean of the output of that specific filter
loss = -torch.mean(self.conv_output)
print('Iteration:', str(i), 'Loss:', "{0:.2f}".format(loss.data.numpy()))
# Backward
loss.backward()
# Update image
optimizer.step()
# Recreate image
self.created_image = recreate_image(self.processed_image)
# Save image every 20 iteration
if i % 10 == 0:
    print(self.created_image.shape)
    im_path = 'generated/ddream_l' + str(self.selected_layer) + \
        '_f' + str(self.selected_filter) + '_iter' + str(i) + '.jpg'
    save_image(self.created_image, im_path)
```

Attention à l'indentation afin que cette partie du traitement fasse parti de la fonction (on aligne « for index, layer in enumerate(self.model): » et « # Loss function is the mean of the output of the selected layer/filter »).

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

7. La fonction `__main__`

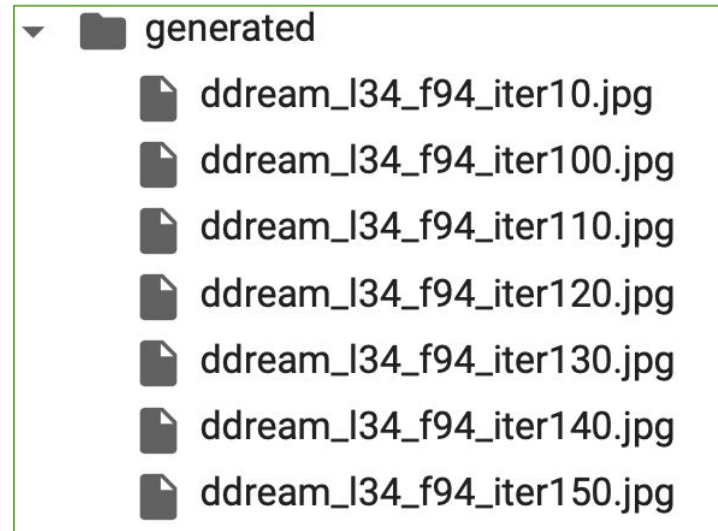
`if __name__ == "__main__"` est la partie qui s'exécute lorsque le script est lancé.

```
if __name__ == '__main__':  
    cnn_layer = 34  
    filter_pos = 94  
    im_path = 'input_images/snake.png'  
    # Fully connected layer is not needed  
    pretrained_model = models.vgg19(pretrained=True).features  
    dd = DeepDream(pretrained_model, cnn_layer, filter_pos, im_path)  
    dd.dream()
```

On exécute le Deep Dream sur l'image d'exemple skane.png .

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

8. Visualisation des résultats



On visualise les images générées dans le dossier « generated ».

Source : <https://github.com/utkuozbulak/pytorch-cnn-visualizations>

Références

- <https://github.com/utkuozbulak/pytorch-cnn-visualizations>