SY32 – TD Appr. Profond 01 : Inférence de réseaux de neurones profonds pré-entraînés

Dans ce TD, nous allons utiliser la bibliothèque `PyTorch` pour utiliser des algorithmes de réseaux de neurones profonds.

Installation

Pour l'installation de `PyTorch`, suivre ou s'inspirer des instructions données dans la page web de l'UV : `https://vision.uv.utc.fr/doku.php?id=setup-python`.

Exercice 1 : Classification d'images avec des architectures existantes et pré-entraînées

On souhaite utiliser le réseau de neurones convolutifs VGG16 du Vision Geometry Group de l'université d'Oxford. VGG est un réseau aujourd'hui classique, ayant eu son heure de gloire en 2014 en étant le meilleur algorithme pour la tâche de classification dans le challenge ILSVRC2014 sur les données ImageNet `https://image-net.org/challenges/LSVRC/2014/results#clsloc`.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

FIGURE 1 – VGG16 est la configuration D. Les activations ReLU ne sont pas mentionnées.

1 . Charger le modèle VGG16 déjà entraîné à l'aide des commandes ci-dessous :

```
import torch
import torchvision.models as models

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

# modele pre-entraine avec ImageNet
model = models.vgg16(pretrained=True).to(device)
```

Note : la première fois que le modèle est chargé, il doit être téléchargé sur la machine ($\approx$ 530 Mo).

2 . Afficher l'architecture du réseau de neurones, deux méthodes sont données ici :

```
print(model)
```

```
from torchinfo import summary
print(summary(model))
```

La comparer avec l'architecture de VGG16 telle que décrite dans l'article d'origine, reprise dans la Figure 1.

3 . Charger l'image `cougar.jpg` du TD-A05 et la préparer pour la traiter avec ce réseau, à l'aide des commandes ci-dessous :

```
import torchvision.transforms as T
from torchvision.io import read_image

# pre-traitements sur les images d'entree
preprocess = T.Compose([T.ConvertImageDtype(torch.float32), # passe dans [0 ;1]
                        T.Resize(256),
                        T.CenterCrop(224), # VGG traite des images 224x224 px
                        # standardisation tq moy=0 et etype=1, sur stats ImageNet
                        # aide le reseau a garder des poids autour de 0
                        T.Normalize(mean=[0.485, 0.456, 0.406],
                                    std=[0.229, 0.224, 0.225])
                        ])

# read_image donne directement un Tensor sans image PIL intermediaire
cougar = read_image('cougar.jpg').to(device)
print(cougar.shape) # 3xHxW
print(cougar.dtype)
img = preprocess(cougar)
print(img.shape)
print(img.dtype)
```

4 . La commande `y = model(input_batch)` traite les images du batch à travers le réseau, jusqu'à un vecteur de 1000 valeurs permettant d'en prédire la classe (le réseau a été entraîné avec ImageNet, contenant 1000 classes différentes). Le réseau implémenté en `pytorch` ne contient pas la couche `softmax` vue en Figure 1. Il faut rajouter manuellement l'opération `softmax` pour extraire le vecteur de scores sur l'ensemble des 1000 classes possibles (pouvant être interprétés comme des probabilités).

```
from torch import nn

model.eval() # comportement pour eval/inference
input_batch = img[None,] # ajout dimension Batch en 1er ; <=> unsqueeze()

with torch.no_grad(): # ou with torch.inference_mode() :
    y = model(input_batch)
    y = nn.functional.softmax(y, dim=1) # softmax apres la derniere couche
    y = y[0] # reprendre la seule image traitee dans cet exemple
print(y.shape)
print(y.sum())
```

5 . Écrire et appliquer la fonction `predictions` pour retrouver les noms des classes ayant les scores les plus élevés.

```
# lire la liste des classes ImageNet
with open("imagenet_classes.txt", "r") as f:
    classes = [s.strip() for s in f.readlines()]

# decoder la classe de l'image
def predictions(y, topn = 5):
    res = "Predictions et probas :\n"
    top_prob, top_cid = torch.topk(y, topn)
    for i in range(top_prob.size(0)):
        res += "{:20}\t{:.6f}\n".format(classes[top_cid[i]], top_prob[i])
    return res
```

6 . Utiliser VGG16 pour prédire les classes des images `crab.jpg` et `kangaroo.jpg`.

7 . La dernière couche du réseau de neurones associée au `softmax` sert à prédire la classe de l'image. Les couches intermédiaires font partie du processus d'extraction de caractéristiques. Pour récupérer la représentation de l'image sur l'avant dernière couche dense (appelée parfois aussi linéaire, ou entièrement connectée), on peut construire le modèle suivant, où l'inférence donnera un dictionnaire pour chaque sortie dérivée (`create_feature_extractor` nécessite `torchvision` ≥ 0.11) :

```
from torchvision.models.feature_extraction import create_feature_extractor

# dictionnaire des couches desirees avec nommage a la carte, ici juste 1
return_nodes = {"classifier.3": "features"} # couche dense avant ReLU
model_feat = create_feature_extractor(model, return_nodes=return_nodes)
model_feat.eval()
```

Avec les versions plus anciennes de `torchvision`, on peut stocker de côté la sortie d'une couche intermédiaire au moment de l'inférence classique du réseau, au lieu de construire un réseau dérivé ; ici dans un dictionnaire `activation` :

```
activation = {}
def get_activation(name):
    def hook(model, input, output):
        activation[name] = output.clone().detach()
    return hook

model.classifier[3].register_forward_hook(get_activation('features'))
```

8 . La commande `x = model_feat(input_batch)['features']`, ou bien `x = activation['features']`, en fonction de l'implémentation écrite, permet alors de récupérer les vecteurs de dimension 4096 représentant les images du batch. Reprendre les images du TD-A05 et utiliser le réseau VGG16 pour trouver, pour chacune des images `cougar.jpg`, `crab.jpg` et `kangaroo.jpg`, les images les plus proches parmi les 300 images de la base (par distance euclidienne entre vecteurs de description).

9 . Reprendre l'exercice en utilisant le réseau `ResNet-50` (modèle pré-entraîné ≈ 98 Mo) ou `EfficientNet-B0` (modèle pré-entraîné ≈ 21 Mo) à la place de `VGG16`.

```
model = models.resnet50(pretrained=True).to(device)
```

```
model = models.efficientnet_b0(pretrained=True).to(device)
```