M1 Deep Learning - TP 4 - LIGHTNING

Université de Caen Normandie

Avril 2023

1 Introduction

L'objectif de ce TP est de vous familiariser avec la librairie PyTorch Lightning [1] en réalisant un réseau de neurones capable de classifier des panneaux routiers. Nous mettons à votre disposition une base de données contenant des images de quatre types de panneaux différents, ainsi que des images pour l'entraînement et le test du modèle.

Le TP se déroulera en quatre étapes. Les trois premières se feront dans un notebook Jupyter et la dernière étape montrera les capacités de PyTorch Lightning à entraîner des modèles depuis une ligne de commande de shell.

Les étapes de ce TP sont les suivantes :

1. Préparer les données et les batches en créant la classe 'GTSRBDataModule' qui hérite de 'pl.LightningDataModule'.
2. Construire un modèle de réseau de neurones et les méthodes nécessaires pour l'entraîner en créant la classe 'MyModel' qui hérite de 'pl.LightningModule'.
3. Entraîner le modèle à l'aide des données préparées dans l'étape 1.
4. Utiliser PyTorch Lightning en ligne de commande pour entraîner le modèle à partir d'un shell.

2 Première étape : définition d'un "Data Module"

Nous allons commencer par travailler sur la préparation des données en utilisant le module `datamodule` [2] de `lightning`. Après avoir lu la documentation et votre cours, définir la classe `GTSRBDataModule` qui hérite de `pl.LightningDataModule` et qui permette de charger les images du *German Traffic Sign Recognition Benchmark* qui vous sont fournies.

Vous générerez des batches de 8 images et gardez 10% des images d'entrainement comme données de validation.

Vous définirez en particulier les méthodes :

— `__init__` : définition de constantes comme le chemin d'accès des données, la taille des batches ou le pourcentage d'images d'entrainement gardées comme données de validation. Ces constantes doivent être passées comme argument de l'init.

— `prepare_data` : préparation des données. Dans notre cas, décompression des archives par la fonction `torchvision.datasets.utils.extract_archive`, si nécéssaire.

1. `https://lightning.ai/`
2. `https://lightning.ai/docs/pytorch/stable/data/datamodule.html`

— `setup` : définition des datasets. La méthode `torchvision.datasets.ImageFolder()` va vous permettre de générer des datasets en leur appliquant des transformations. Nous vous demandons de ramener les images à une taille de 224x224 et d'appliquer une rotation aléatoire de 10 degrés (data augmentation) aux données d'entrainement).
Regardez dans la documentation le rôle de l'argument 'stage' et utilisez-le à bon escient. Cette variable pourra prendre les valeurs 'None', 'test' ou 'fit'.
— `train_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de train.
— `val_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de validation
— `test_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de test
— `predict_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de test

Une fois cette classe définie, vous devez être capable de générer et visualiser les batches d'images des jeux d'entrainement, de validation et de test au moyen de ces lignes de commandes :

```
import matplotlib.pyplot as plt
gtsrb = GTSRBDataModule()
gtsrb.prepare_data()
gtsrb.setup()
d = next(iter(gtsrb.train_dataloader()))
plt.imshow(d[0][0,0],cmap='gray')
plt.show()
```

3 Seconde étape : construction du réseau de classification

Dans une nouvelle cellule de calcul, définir la classe `MyLightningModuleNet` qui hérite de `lightning.LightningModule`, qui permette de construire le modèle de réseau de neurones et les différentes fonctions permettant de l'entrainer.

La documentation en ligne se trouve ici : `https://lightning.ai/docs/pytorch/stable/common/lightning_module.html`.

Le réseau de classification que nous vous demandons d'utiliser est un réseau de type 'resnet18' que vous utiliserez pré-entrainé pour une tâche de classification de type ImageNet.

Récupérer le réseau et ses poids peut se faire via :
`resnet18(weights=ResNet18_Weights.DEFAULT)`.

Ce classifieur possède 1000 sorties car la classification ImageNet est une classification à 1000 classes. En regardant les données qui vous sont fournies, vous noterez que, dans le TP, il s'agit d'un problème de classification à 4 classes. Vous remplacerez donc la couche de classification du réseau chargé par votre propre couche de classification (qui est /tt self.resnet18.fc), qui sera une couche entièrement connectée.

Vous devrez en particulier définir les méthodes :
— `__init__` : définition des différentes couches du réseau et des constantes
— `forward` : calcule la sortie du réseau en fonction de son entrée
— `training_step` : retourne la loss correspondant à un batch donné
— `validation_step` : effectue le traitement d'un batch sur les données de validation
— `test_step` : effectue le traitement d'un batch sur les données de test

— `configure_optimizers` : retourne l'optimiseur choisi pour entrainer le réseau

4 Troisième étape : entrainement du réseau de neurones

Une fois les classes précédentes définies, vous êtes capable d'entrainer et d'évaluer le modèle grâce aux lignes de code suivantes :

```
trainer = pl.Trainer(max_epochs=4)
model = MyLightningModuleNet()
trainer.fit(model, train_dataloaders=gtsrb.train_dataloader(),
    val_dataloaders=gtsrb.val_dataloader())
trainer.validate(model, dataloaders=gtsrb.test_dataloader())
```

5 Dernière étape : entrainement du réseau de neurones depuis une console shell

Télécharger votre notebook comme fichier python et ne gardez que le code qui se trouve dans les classes, et ajouter à la fin du fichier les lignes de code suivantes :

```
from lightning.pytorch.cli import LightningCLI

from my_model import MyModel
from my_dataset import *

def cli_main():
    cli = LightningCLI(MyModel, GTSRBDataModule)

if __name__ == "__main__":
    cli_main()
```

Vous pouvez désormais entrainer le modèle par une ligne de commande de type :

```
python3 tp4.py fit --trainer.max_epochs 2
```

ce qui est pratique lorsque l'on veut entrainer des modèles sur des serveurs. Les hyper paramètres peuvent ainsi être passés par la ligne de commandes et tous les résultats sont 'loggés'.