⚙ Présentation Hydra dans Lightning-Hydra-Template

🔍 Introduction à Hydra

- Hydra est un **framework de configuration** développé par **Facebook Research**.
- Utilisé dans le **template Lightning-Hydra-Template** pour :
  - Gérer des **configurations hiérarchiques**.
  - Modifier dynamiquement via YAML ou CLI.
- Hydra va instancier dynamiquement les classes cibles dans le train.py avec les fichiers de config qui contiennent les variables.

🗂️ Structure des Configurations

configs/

├── common # Folder with many config (hydra, trainer and paths in config_base.yaml)
├── callbacks/ # Callbacks
├── logger/ # Loggers
├── data/ # Datasets
├── experiment/ # Expériences
├── model/ # Modèles
├── Tuto/ # tuto pour hydra
├── train.yaml # Entraînement principal
└── eval.yaml # Évaluation principale

Les configurations sont organisées dans le dossier configs/ avec différents sous-dossiers pour chaque type de composant
(callbacks, loggers, modèles, etc.).
Le fichier principal est configs/train.yaml qui définit la configuration par défaut pour l'entraînement. train.yaml:5-10

⚙️ Comment Modifier les Variables de Configuration

On peut modifier les valeurs de ses variables de plusieurs façons :

1. Modification Directe dans les Fichiers YAML

Vous pouvez modifier directement les variables dans les fichiers YAML. Par exemple, vous pouvez changer le nombre d'epoch max dans le fichier de configuration `configs/common/config_base.yaml` :

```yaml
trainer:
  _target_: lightning.pytorch.trainer.Trainer
  default_root_dir: ${paths.output_dir}
  min_epochs: 1
  max_epochs: 10
  accelerator: gpu
  devices: 1
  check_val_every_n_epoch: 1
  deterministic: false
```

2. Surcharge depuis la Ligne de Commande

Modifiez un ou plusieurs paramètres sans toucher les fichiers YAML :

```
python src/train.py model.optimizer.lr=0.001 data.batch_size=64
```

3. Configurations Alternatives

Utiliser un modèle ou logger différent (il faut bien regarder dans le dossier config/model les modeles disponible et pour les loggers dans config/common/logger)

```
python src/train.py model=autre_modele common/logger@logger=nom_logger
```

ATTENTION , pour surcharger le logger et les callbacks les commandes sont:

```
python src/train.py common/logger@logger=nom_logger
```

```
python src/train.py common/callbacks@callbacks=nom_callbacks
```

🧪 Configurations d'Expériences

Objectif:

Définir une configuration complète d'expérience dans un seul fichier YAML (ce qui évite de tout rechanger pour revoir une config) :

- Modèle
- Dataset
- Callbacks
- Nombre d’époch
- Seed

Voici un exemple : `configs/experiment/mon_experience.yaml`

```yaml
# @package _global_
defaults:
  - override /data: mnist
  - override /model: mnist
  - override /common/callbacks@callbacks: default
  - override /common/logger@logger: mlflow

# Paramètres spécifiques à cette expérience
tags: ["mnist"]

# Surcharger des paramètres spécifiques
trainer:
  max_epochs: 10
  min_epochs: 5
  gradient_clip_val: 0.5
  accelerator: cpu
  devices: 1

model:
  optimizer:
    lr: 0.002
  net:
    lin1_size: 128
    lin2_size: 256
    lin3_size: 64

data:
  batch_size: 64
```

Puis vous pouvez utiliser la commande :

```
    python src/train.py experiment=mon_experience
```

Dans cette exemple, cela charge :

-Le dataset MNIST

-Le modèle MNIST

-Les callbacks par défaut

-La seed 12345

-20 epochs

-Un taux d’apprentissage de 0.002

Donc si vous voulez surcharger des paramètres dans une experience, regardez d'abord le fichier de config original pour voir la structure des variables à surcharger.

✅ Vérification finale

Si tout a bien fonctionné...

Vous devriez voir l’image suivante générée au début de l'entraînement dans le terminal: