Code review

General overview ◂

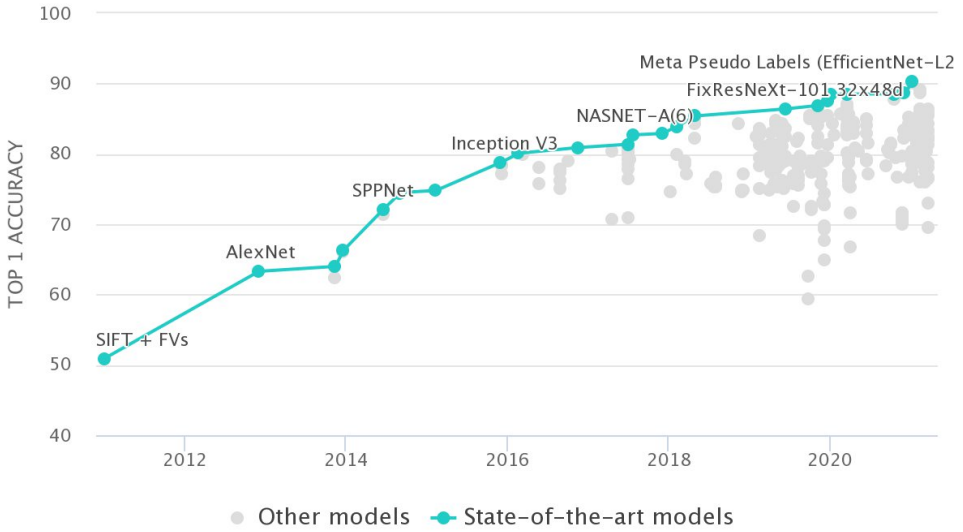
Detailed overview ◂

Data – ImageNet

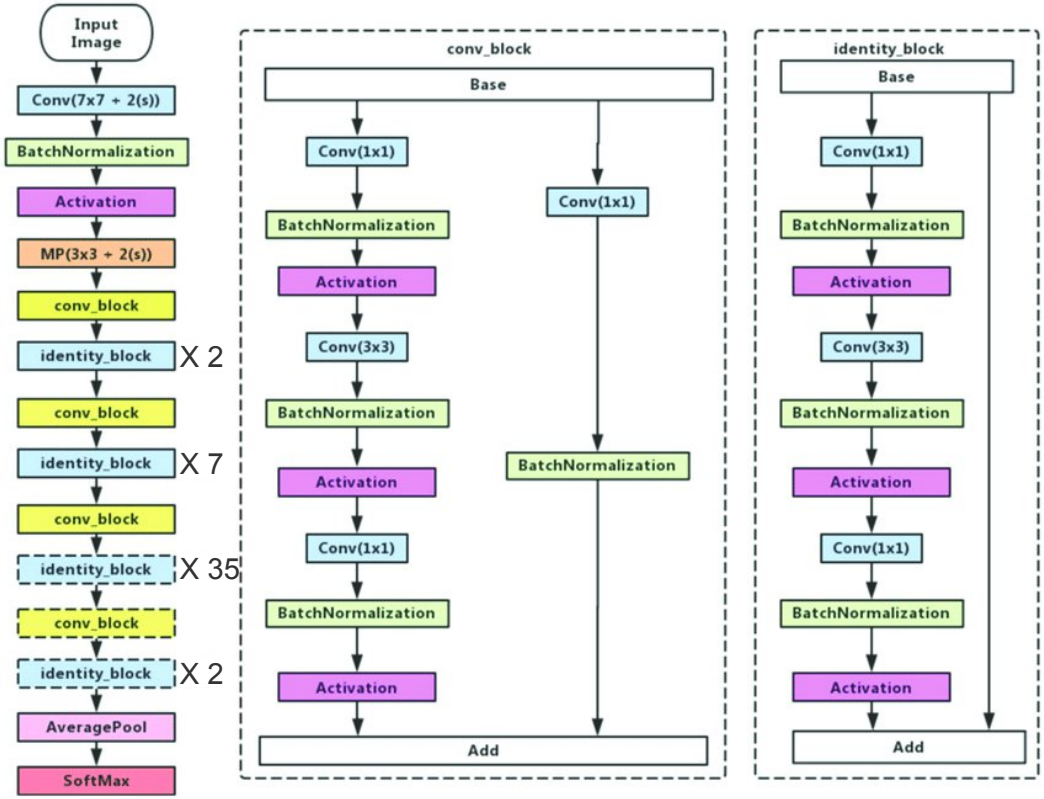
Goal:
Classification (1000 classes)

Dataset:
Train dataset: **1.2 M** labeled images
Validation dataset: **50,000** labeled images
https://www.image-net.org/





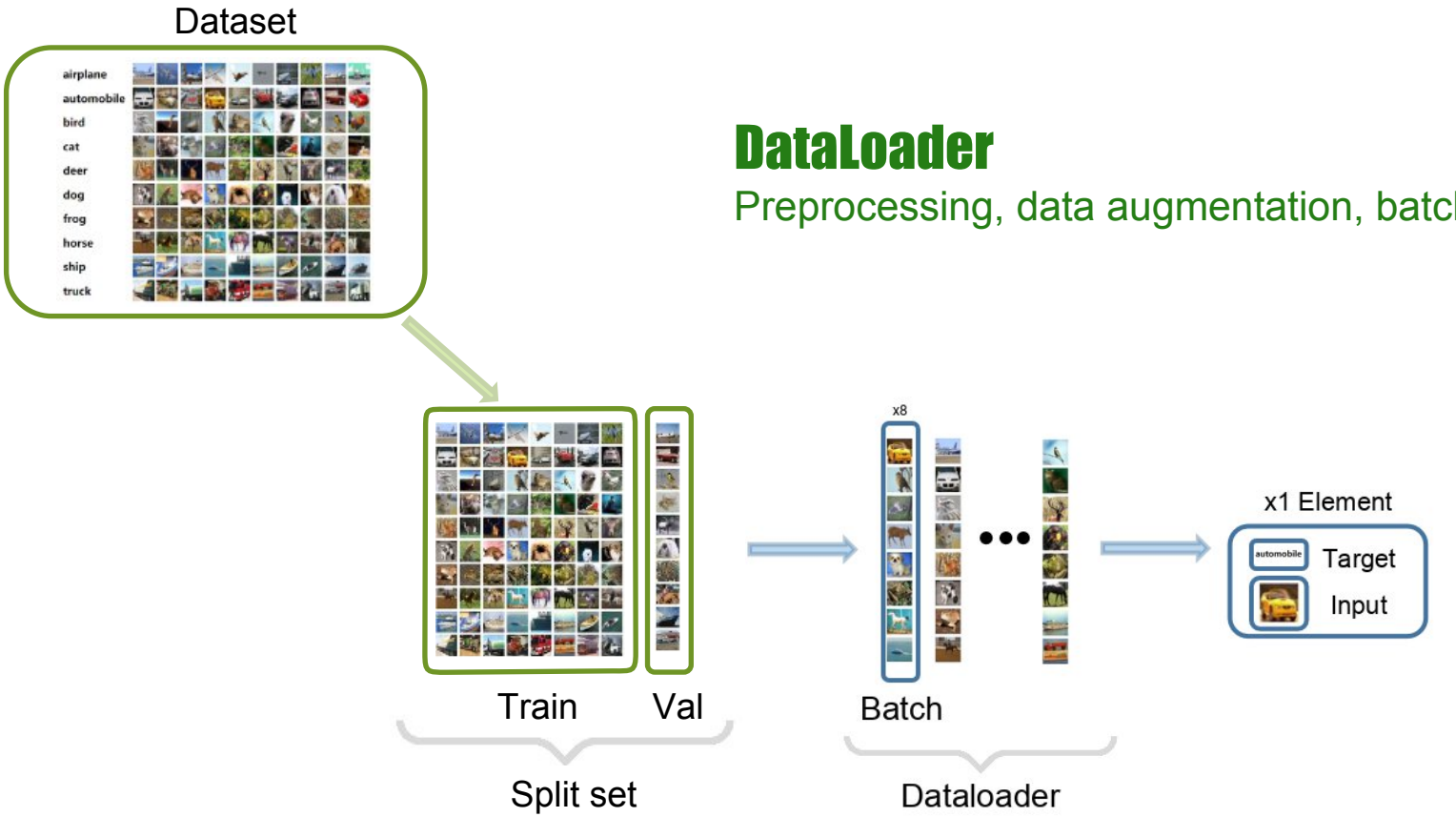
ImageNet – ResNet-



ResNet-152

- Residual (skip) connections
- Bottleneck blocks
  - 1×1 → 3×3 → 1×1 convolutions
- 3 + 8 + 36 + 3 blocks
  - ≈ 60 M parameters, ≈ 11.5 GMACs per 224×224 image

Training Loop – DataLoader



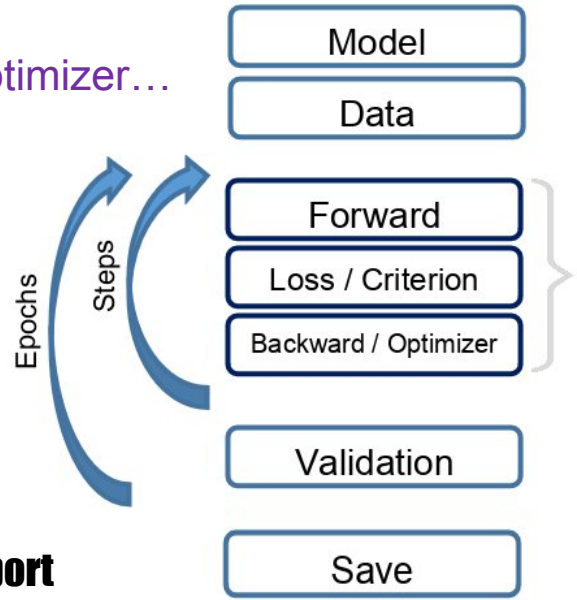
DataLoader

Preprocessing, data augmentation, batching…

Training Loop

Instantiation

Model, distribution, optimizer…



DataLoader

Preprocessing, data augmentation, batching…

Training

Mixed precision, distribution…

Validation

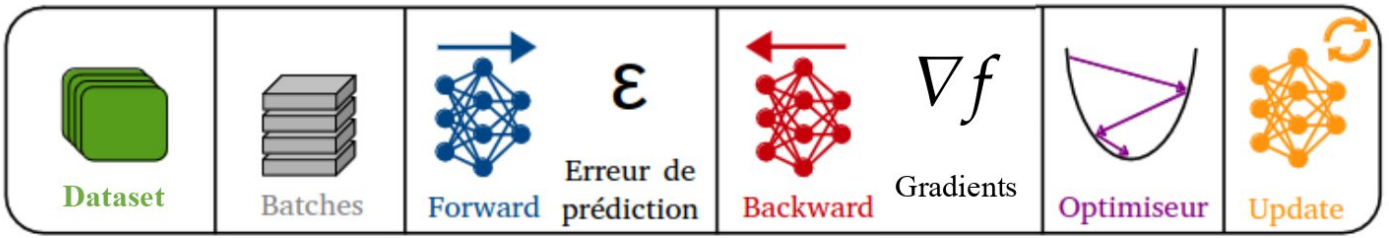
Mixed precision, distribution…

Checkpoint & report

Training step – GPU acceleration

DataLoader
Preprocessing, batching…

Training
Mixed precision, distribution, metrics…



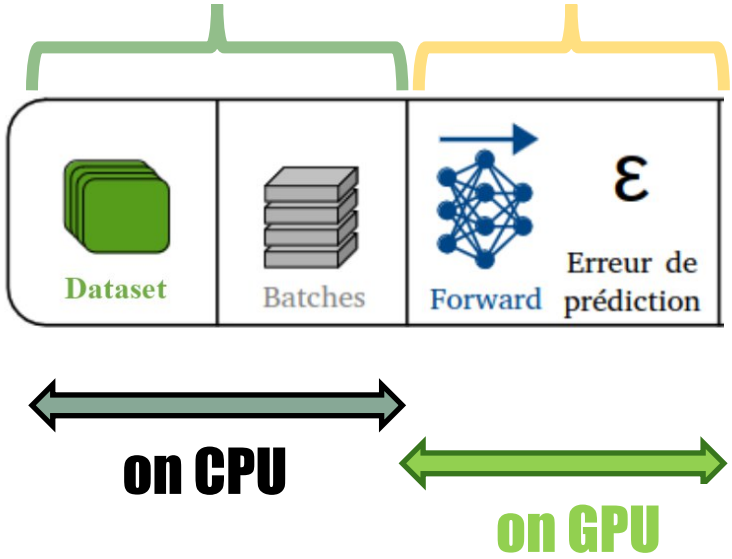
on CPU

on GPU

Validation step – GPU acceleration

DataLoader
Preprocessing, batching…

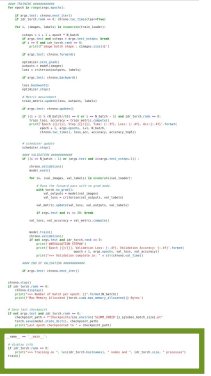
Validation
Mixed precision, distribution, metrics…



dlojz.py – Import & run







```
import os
import contextlib
import argparse
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
from torch.utils.checkpoint import checkpoint_sequential
import torch
import numpy as np
import apex

import idr_torch
from dlojz_chrono import Chronometer
from torchmetrics.aggregation import MeanMetric
from torchmetrics.classification import MulticlassAccuracy

import random
random.seed(123)
np.random.seed(123)
torch.manual_seed(123)
```

reproducibility

Imports: os, argparse, PyTorch, torchvision…

PyTorch NumPy torchvision

Chronometer (DLO-JZ)

Time logging & in-house profiler



```
if __name__ == '__main__':

    # display info
    if idr_torch.rank == 0:
        print(">>> Training on ", len(idr_torch.hostnames), " nodes and ", idr_torch.size, " processes")
    train()
```

idr_torch (Jean Zay)

Distribution helpers: rank, size, nodes

dlojz.py – Argument parser







```python
## import ... ## Add here the libraries to import

VAL_BATCH_SIZE=256

#*****************************************************************************************************************
def train():                                                                                                    #
    parser = argparse.ArgumentParser()                                                                          #
    parser.add_argument('-b', '--batch-size', default=128, type =int,                                           #
                        help='batch size per GPU')                                                              #
    parser.add_argument('-e','--epochs', default=1, type=int,                                                   #
                        help='number of total epochs to run')                                                   #
    parser.add_argument('--image-size', default=224, type=int,                                                  #
                        help='Image size')                                                                      #
    parser.add_argument('--lr', default=0.1, type=float,                                                        #
                        help='learning rate')                                                                   #
    parser.add_argument('--wd', default=0., type=float,                                                         #
                        help='weight decay')                                                                    #
    parser.add_argument('--mom', default=0.9, type=float,                                                       #
                        help='momentum')                                                                        #
    parser.add_argument('--test', default=False, action='store_true',     ##    DON'T MODIFY    ########      #
                        help='Test 50 iterations')                                                              #
    parser.add_argument('--test-nsteps', default='50', type=int,                                                #
                        help='the number of steps in test mode')                                                #
    parser.add_argument('--num-workers', default=10, type=int,                                                  #
                        help='num workers in dataloader')                                                       #
    parser.add_argument('--persistent-workers', default=True, action=argparse.BooleanOptionalAction,           #
                        help='activate persistent workers in dataloader')                                       #
    parser.add_argument('--pin-memory', default=True, action=argparse.BooleanOptionalAction,                    #
                        help='activate pin memory option in dataloader')                                        #
    parser.add_argument('--non-blocking', default=True, action=argparse.BooleanOptionalAction,                  #
                        help='activate asynchronuous GPU transfer')                                             #
    parser.add_argument('--prefetch-factor', default=3, type=int,                                               #
                        help='prefectch factor in dataloader')                                                  #
    parser.add_argument('--drop-last', default=False, action=argparse.BooleanOptionalAction,                    #
                        help='activate drop_last option in dataloader')                                         #
#*****************************************************************************************************************

    ## Add parser arguments

    args = parser.parse_args()
```

Configurable arguments:

`--batch-size` : batch size per GPU
`--epochs` : number of epochs
`--image-size` : image size

Optimizer:
`--lr` : learning rate
`--wd` : weight decay
`--mom` : momentum

Test mode:
`--test` : test mode
`--test-nsteps` : number of steps in test mode

DataLoader optimization:
`--num-workers`
`--persistent-workers`
`--pin-memory`
`--non-blocking`
`--prefetch-factor`
`--drop-last`

dlojz.py – Instantiation

```python
## chronometer initialisation
chrono = Chronometer()

# define model
model = models.resnet152()
archi_model = 'Resnet-152'

if idr_torch.rank == 0: print(f'model: {archi_model}')
if idr_torch.rank == 0: print('number of parameters: {}'.format(sum([p.numel()
                                        for p in model.parameters()])))
# distribute batch size (mini-batch)
num_replica = idr_torch.size
mini_batch_size = args.batch_size
global_batch_size = mini_batch_size * num_replica

if idr_torch.rank == 0:
    print(f'global batch size: {global_batch_size} - mini batch size: {mini_batch_size}')

# define loss function (criterion) and optimizer
criterion = torch.nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = torch.optim.SGD(model.parameters(), args.lr, momentum=args.mom, weight_decay=args.wd)

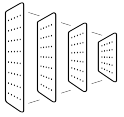
if idr_torch.rank == 0: print(f'Optimizer: {optimizer}')

# define metrics
train_metric = {}
train_metric['loss'] = MeanMetric()
train_metric['acc'] = MulticlassAccuracy(num_classes=1000, average='micro')
val_metric = {}
val_metric['loss'] = MeanMetric()
val_metric['acc'] = MulticlassAccuracy(num_classes=1000, average='micro')
```

```python
#LR scheduler to accelerate the training time
scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=args.lr,
                                                steps_per_epoch=N_batch, epochs=args.epochs)
```



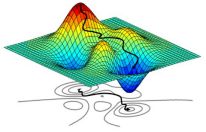
Chronometer



Model: ResNet-152

mini batch size ⟺ global batch size

$\mathcal{E}$ CrossEntropyLoss



SGD optimizer



Metrics



LR scheduler

dlojz.py – DataLoader



```
######### DATALOADER ############
# Define a transform to pre-process the training images.

if idr_torch.rank == 0: print(f"DATALOADER {args.num_workers} {args.persistent_workers} {args.pin_memory}")

transform = transforms.Compose([
        transforms.RandomResizedCrop(args.image_size),  # Random resize - Data Augmentation
        transforms.RandomHorizontalFlip(),              # Horizontal Flip - Data Augmentation
        transforms.ToTensor(),                          # convert the PIL Image to a tensor
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))
        ])


train_dataset = torchvision.datasets.ImageNet(root=os.environ['ALL_CCFRSCRATCH']+'/imagenet',
                                              transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=mini_batch_size,
                                           shuffle=True,
                                           num_workers=args.num_workers,
                                           persistent_workers=args.persistent_workers,
                                           pin_memory=args.pin_memory,
                                           prefetch_factor=args.prefetch_factor,
                                           drop_last=args.drop_last)


val_transform = transforms.Compose([
          transforms.Resize((256, 256)),
          transforms.CenterCrop(224),
          transforms.ToTensor(),    # convert the PIL Image to a tensor
          transforms.Normalize(mean=(0.485, 0.456, 0.406),
                               std=(0.229, 0.224, 0.225))])

val_dataset = torchvision.datasets.ImageNet(root=os.environ['ALL_CCFRSCRATCH']+'/imagenet', split='val',
                    transform=val_transform)

val_loader = torch.utils.data.DataLoader(dataset=val_dataset,
                                         batch_size=VAL_BATCH_SIZE,
                                         shuffle=False,
                                         num_workers=args.num_workers,
                                         persistent_workers=args.persistent_workers,
                                         pin_memory=args.pin_memory,
                                         prefetch_factor=args.prefetch_factor,
                                         drop_last=args.drop_last)


N_batch = len(train_loader)
N_val_batch = len(val_loader)
N_val = len(val_dataset)
```

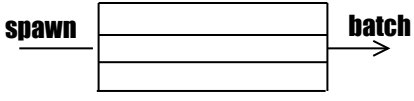
Train dataset:

RandomResizedCrop
RandomHorizontalFlip
+ Normalize



Shuffling

DataLoader optimization



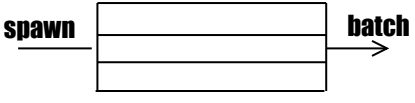
Validation dataset:

Resize
CenterCrop
+ Normalize



No shuffling

DataLoader optimization



dlojz.py – Training

```
chrono.start()

#### TRAINING ############
for epoch in range(args.epochs):

    if args.test: chrono.next_iter()
    if idr_torch.rank == 0: chrono.tac_time(clear=True)

    for i, (images, labels) in enumerate(train_loader):

        csteps = i + 1 + epoch * N_batch
        if args.test and csteps > args.test_nsteps: break
        if i == 0 and idr_torch.rank == 0:
            print(f'image batch shape : {images.size()}')

        if args.test: chrono.forward()

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)

        if args.test: chrono.backward()

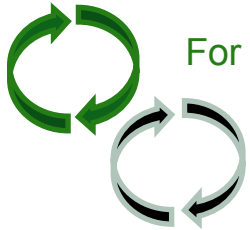
        loss.backward()
        optimizer.step()

        # Metric mesurement
        train_metric['loss'].update(loss)
        train_metric['acc'].update(outputs, labels)

        if args.test: chrono.update()

        if ((i + 1) % (N_batch//10) == 0 or i == N_batch - 1):
            train_loss, accuracy = train_metric['loss'].compute(), train_metric['acc'].compute()
            train_metric['loss'].reset(), train_metric['acc'].reset()
            if idr_torch.rank == 0:
                print('Epoch [{}/{}], Step [{}/{}], Time: {:.3f}, Loss: {:.4f}, Acc:{:.4f}'.format(
                      epoch + 1, args.epochs, i+1, N_batch,
                      chrono.tac_time(), train_loss, accuracy))


        # scheduler update
        scheduler.step()
```

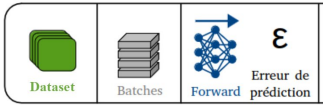


For each epoch

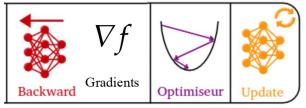
For each *batch*

test mode: 50 steps

```
optimizer.zero_grad()
outputs = model(images)
loss = criterion(outputs, labels)
```




```
loss.backward()
optimizer.step()
```





Accumulate metrics (loss, accuracy)
Compute and print them 10× per epoch



LR scheduler step (every batch)

dlojz.py – Validation



```
#### VALIDATION #############
if ((i == N_batch - 1) or (args.test and i==args.test_nsteps-1)) :

    chrono.validation()
    model.eval()

    for iv, (val_images, val_labels) in enumerate(val_loader):

        # Runs the forward pass with no grad mode.
        with torch.no_grad():
            val_outputs = model(val_images)
            val_loss = criterion(val_outputs, val_labels)

        val_metric['loss'].update(val_loss)
        val_metric['acc'].update(val_outputs, val_labels)

        if args.test and iv >= 20: break

    val_loss, val_accuracy = val_metric['loss'].compute(), val_metric['acc'].compute()
    val_metric['loss'].reset(), val_metric['acc'].reset()

    model.train()
    chrono.validation()
    if not args.test and idr_torch.rank == 0:
        print('##EVALUATION STEP##')
        print('Epoch [{}/{}], Validation Loss: {:.4f}, Validation Accuracy: {:.4f}'.format(
                                epoch + 1, args.epochs, val_loss, val_accuracy))
        print(">>> Validation complete in: " + str(chrono.val_time))

#### END OF VALIDATION #############
```

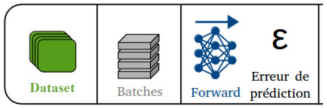


After each epoch
(or at the end of test mode)



For each *validation batch*
(test mode: 20 steps)

```
# Runs the forward pass with no grad mode.
with torch.no_grad():
    val_outputs = model(val_images)
    loss = criterion(val_outputs, val_labels)
```





Accumulate metrics (loss, accuracy)

dlojz.py – Checkpoint & Report

```
chrono.stop()
if idr_torch.rank == 0:
    chrono.display()
    print(">>> Number of batch per epoch: {}".format(N_batch))
    print(f'Max Memory Allocated {torch.cuda.max_memory_allocated()} Bytes')


# Save last checkpoint
if not args.test and idr_torch.rank == 0:
    checkpoint_path = f"checkpoints/{os.environ['SLURM_JOBID']}_{global_batch_size}.pt"
    torch.save(model.state_dict(), checkpoint_path)
    print("Last epoch checkpointed to " + checkpoint_path)
```

Report: timings + max GPU memory

Checkpoint saved at the end of training (not in test mode)



Trained model weights

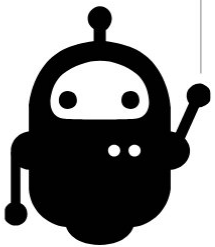
TP0: Set up your environment



- Open a terminal and clone the repository

```
local:~$ ssh jean-zay

jz:~$ cd $WORK
jz:~$ git clone https://github.com/idriscnrs/dlojz_tp/
```



- Open Firefox
- Go to jupyterhub.idris.fr

TP0: JupyterHub

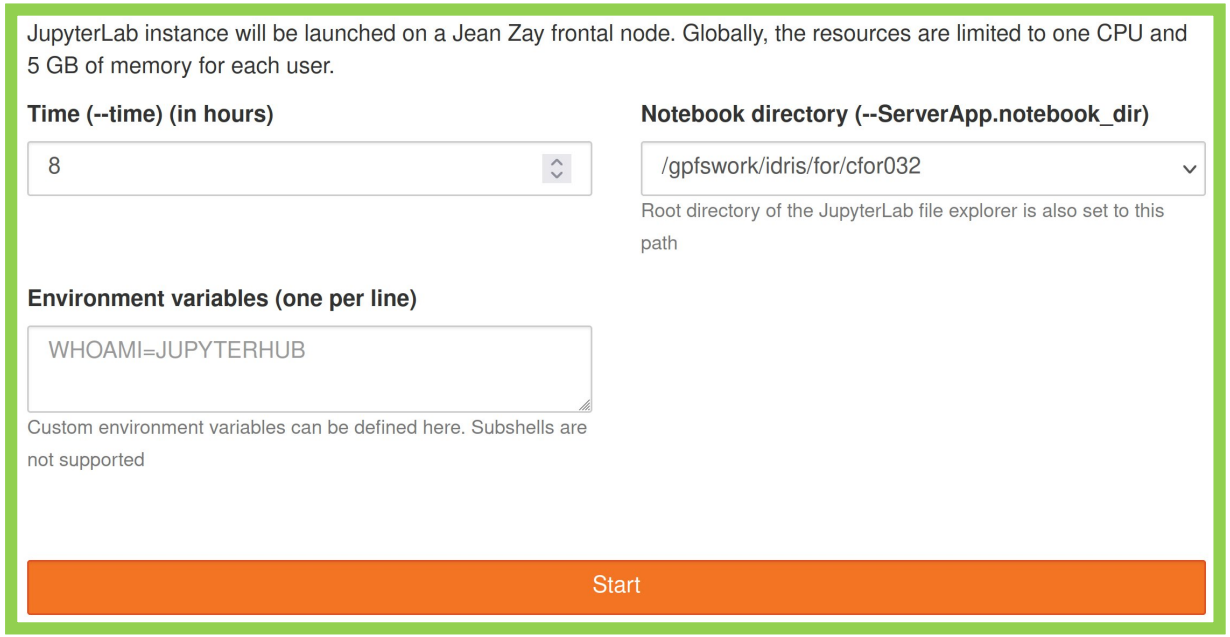
- Log in with your training account
- Add a new instance



- Select the spawner ‘Interactive’



- Fill in the configuration



- Start

TP0: Notebook



- Open the notebook DLO-JZ_Jour1.ipynb
- Select the kernel pytorch-gpu/py3/2.8.0 (top right) if it is not detected
- Choose a pseudonym
- Submit a first job
- Explore the reference script and its options