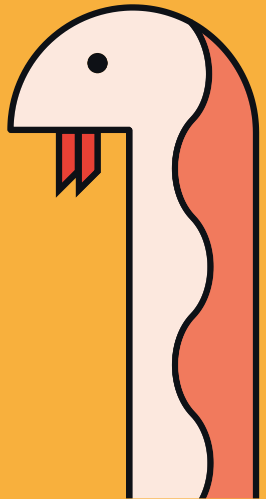
OROBIX

PYCON(IT) 23

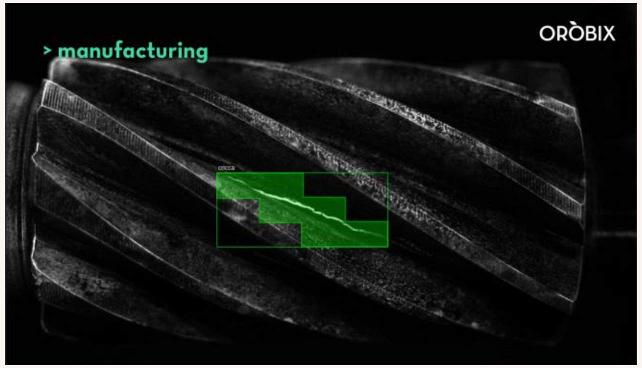
Controlling the Chaos: Reproducible Deep Learning Experiments Using Lightning and Hydra

Refik Can MALLI



Orobix

- **We are 45 People**
  - 10 working in software development
  - 17 working in data science
- **Young team with average of 33 years old**
- **We are located at:**
  - Bergamo
  - Brescia
- **We are applying AI to**
  - Manufacturing
  - Healthcare
  - Games (Reinforcement Learning)



The Happy Data Scientist's Checklist

- Python Ecosystem Integrated Tools
- High-Level API for computations
- Hardware Acceleration
- Production Models
- Data Handling
- Distributed Training
- Logging, Tracking, Checkpointing
- Visualization, Performance Metrics
- Configuring Hyperparameters

PyTorch

- **Ease of use:** It offers a user-friendly interface and a simple and intuitive API
- **Dynamic computational graph:** It allows for flexible and efficient model building and automatic gradient computation.
- **Research focus:** It was developed by researchers for researchers and is widely used in the academic community.
- **Flexible deployment options:** TorchScript, ONNX, TorchServe, C++ Frontend

PyTorch

Data Loading

```python
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root="./data", train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root="./data", train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)

classes = ("plane", "car", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck")
```

PyTorch

Model Building

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Training

```python
for epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader):
        # data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Saving

```python
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
torch.onnx.export(net, inputs, "cifar_net.onnx")
```

PyTorch Lightning

- **Extends PyTorch:** Built top of PyTorch.
- **Less Boilerplate Code:** Handles most of the engineering part.
- **Hardware Acceleration:** GPUs/TPUs/HPUs without code changes.
- **Easy Distributed Training:** Mostly single line setup for different distributed backends.
- **Ready-to-use Toolbox:** Callbacks, Loggers, Profilers and more...

PyTorch Lightning – Data Modules

```python
import torch
from torchvision import datasets, transforms
from pytorch_lightning import LightningDataModule

class CIFAR10DataModule(LightningDataModule):
    def __init__(self, batch_size=4):
        super().__init__()
        self.batch_size = batch_size
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])

    def prepare_data(self):
        datasets.CIFAR10(root='./data', train=True, download=True)
        datasets.CIFAR10(root='./data', train=False, download=True)

    def setup(self, stage=None):
        self.trainset = datasets.CIFAR10(root='./data', train=True, transform=self.transform)
        self.testset = datasets.CIFAR10(root='./data', train=False, transform=self.transform)

    def train_dataloader(self):
        return torch.utils.data.DataLoader(self.trainset, batch_size=self.batch_size, shuffle=True, num_workers=2)

    def val_dataloader(self):
        return torch.utils.data.DataLoader(self.testset, batch_size=self.batch_size, shuffle=False, num_workers=2)
```

Runs on main process

Runs on every process

Used for training

Used for validation

PyTorch Lightning - Training

```python
class CNNModel(LightningModule):
    def __init__(self, learning_rate, momentum):
        super().__init__()
        self.save_hyperparameters()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def training_step(self, batch, batch_idx):
        inputs, targets = batch
        outputs = self(inputs)
        loss = F.cross_entropy(outputs, targets)
        self.log("train_loss", loss)
        return loss

    def configure_optimizers(self):
        optimizer = torch.optim.SGD(
            self.parameters(),
            lr=self.hparams.learning_rate,
            momentum=self.hparams.momentum,
        )
        scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
        return [optimizer], [scheduler]
```

Single Step

Optimizer and Scheduler

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--batch_size", type=int, default=4)
    parser.add_argument("--learning_rate", type=float, default=0.001)
    parser.add_argument("--momentum", type=float, default=0.9)
    parser.add_argument("--max_epochs", type=int, default=2)
    parser.add_argument("--gpus", type=int, default=None)
    parser.add_argument("--accelerator", type=str, default=None)
    parser.add_argument("--patience", type=int, default=3)
    parser.add_argument("--min_delta", type=float, default=0.01)
    parser.add_argument("--log_dir", type=str, default="./logs")
    args = parser.parse_args()

    model = CNNModel(args.learning_rate, args.momentum)
    data_module = CIFAR10DataModule(args.batch_size)
    early_stopping = callbacks.EarlyStopping(
        monitor="train_loss",
        min_delta=args.min_delta,
        patience=args.patience,
        verbose=True,
    )
    logger = CSVLogger(args.log_dir, name="cifar10_logs")
    trainer = Trainer(
        max_epochs=args.max_epochs,
        gpus=args.gpus,
        accelerator=args.accelerator,
        callbacks=[early_stopping],
        logger=logger,
    )
    trainer.fit(model, datamodule=data_module)
```

Where the Chaos begins

```
python main.py \
        --batch_size 8 \
        --learning_rate 0.01 \
        --momentum 0.8 \
        --max_epochs 10 \
        --gpus 1 \
        --accelerator 'gpu' \
        --patience 3 \
        --min_delta 0.01 \
        --log_dir './logs'
```

Where the Chaos begins

```
python main.py \
        --batch_size 8 \
        --learning_rate 0.01 \
        --momentum 0.8 \
        --max_epochs 10 \
        --gpus 1 \
        --accelerator 'gpu' \
        --patience 3 \
        --min_delta 0.01 \
        --log_dir './logs'
```

What if we want to:

- Add another model?
- Check if augmentation helps?
- Try different hyperparameters?
- Integrate different optimizer?
- Try with different dataset?
- Run many different configurations?

Where the Chaos begins

```python
parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
parser.add_argument('data', metavar='DIR', nargs='?', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                         'batch size of all GPUs on the current node when '
                         'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int,
                    help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--dist-backend', default='nccl', type=str,
                    help='distributed backend')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing-distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                         'N processes per node, which has N GPUs. This is the '
                         'fastest way to use PyTorch for either single node or '
                         'multi node data parallel training')
parser.add_argument('--dummy', action='store_true', help="use fake data to benchmark")
```

- When the project becomes complex, argument list gets out of control.
- Classes should be instantiated based on arguments. (string -> class)
- Some arguments might be set up in a group.
- **Dataclasses** can help but do not solve all problems.

Source: https://github.com/pytorch/examples/blob/main/imagenet/main.py

Hydra

- **Configuration composition:** Structure your configurations into hierarchical YAML files. Mix and match configuration files.
- **Defaults and overrides:** easy management of default values and allows for overriding them from the command line.
- **Dynamic configurations:** Setting variables from environment variables, other configuration files or sub-folders.
- **Plugin system:** Launchers (Joblib, Submitit, Sequential), Sweepers (Optuna, Nevergrad)

Hydra

```yaml
# conf/config.yaml
defaults:
  - model: default
  - data: default
  - trainer: default

# trainer/default.yaml
max_epochs: 10
gpus: 1
accelerator: "gpu"
patience: 3
min_delta: 0.01
log_dir: "./logs"

# trainer/cpu.yaml
max_epochs: 2
accelerator: "cpu"
patience: 3
min_delta: 0.01
log_dir: "./logs"

# model/default.yaml
learning_rate: 0.001
momentum: 0.9

# data/default.yaml
batch_size: 32
```

```python
@hydra.main(config_path="conf", config_name="config")
def cli_main(cfg: DictConfig) -> None:
    model = CNNModel(cfg.model.learning_rate, cfg.model.momentum)
    data_module = CIFAR10DataModule(cfg.data.batch_size)
    early_stopping = callbacks.EarlyStopping(
        monitor='train_loss',
        min_delta=cfg.trainer.min_delta,
        patience=cfg.trainer.patience,
        verbose=True
    )
    logger = CSVLogger(cfg.trainer.log_dir, name="cifar10_logs")
    trainer = Trainer(max_epochs=cfg.trainer.max_epochs,
                      gpus=cfg.trainer.gpus,
                      accelerator=cfg.trainer.accelerator,
                      callbacks=[early_stopping],
                      logger=logger)
    trainer.fit(model, datamodule=data_module)

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

Hydra

```yaml
# conf/config.yaml
defaults:
  - model: default
  - data: default
  - trainer: default

# trainer/default.yaml
max_epochs: 10
gpus: 1
accelerator: "gpu"
patience: 3
min_delta: 0.01
log_dir: "./logs"

# trainer/cpu.yaml
max_epochs: 2
accelerator: "cpu"
patience: 3
min_delta: 0.01
log_dir: "./logs"

# model/default.yaml
learning_rate: 0.001
momentum: 0.9

# data/default.yaml
batch_size: 32
```

```python
@hydra.main(config_path="conf", config_name="config")
def cli_main(cfg: DictConfig) -> None:
    model = CNNModel(cfg.model.learning_rate, cfg.model.momentum)
    data_module = CIFAR10DataModule(cfg.data.batch_size)
    early_stopping = callbacks.EarlyStopping(
        monitor='train_loss',
        min_delta=cfg.trainer.min_delta,
        patience=cfg.trainer.patience,
        verbose=True
    )
    logger = CSVLogger(cfg.trainer.log_dir, name="cifar10_logs")
    trainer = Trainer(max_epochs=cfg.trainer.max_epochs,
                      gpus=cfg.trainer.gpus,
                      accelerator=cfg.trainer.accelerator,
                      callbacks=[early_stopping],
                      logger=logger)
    trainer.fit(model, datamodule=data_module)

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

```bash
python main.py trainer.max_epochs=5
python main.py trainer=default,cpu model.learning_rate=0.01,0.1,0.001 --multirun
```

Hydra

Instantiate the class

```yaml
# optimizer/adam.yaml
_target_: torch.optim.Adam
lr: 0.001
betas: [0.9, 0.999]
eps: 1e-08
weight_decay: 0
```

```python
param_list = model.parameters()
optimizer = hydra.utils.instantiate(config.optimizer, param_list)
```

Resolve configuration

```python
def as_tuple(*args: Any) -> Tuple[Any, ...]:
    """Resolves a list of arguments to a tuple."""
    return tuple(args)
OmegaConf.register_new_resolver("as_tuple", as_tuple)
```

Access Environment Variables

```yaml
tracking_uri: ${oc.env:MLFLOW_TRACKING_URI}
data_path: ${oc.env:HOME}/.quadra/datasets/MNIST
```

Checklist Revisited ...

- Python Ecosystem Integrated Tools
- High-Level API for computations
- Hardware Acceleration
- Production Models
- Data Handling
- Distributed Training
- Losses, Performance Metrics
- Logging, Tracking, Checkpointing
- Configuring Hyperparameters

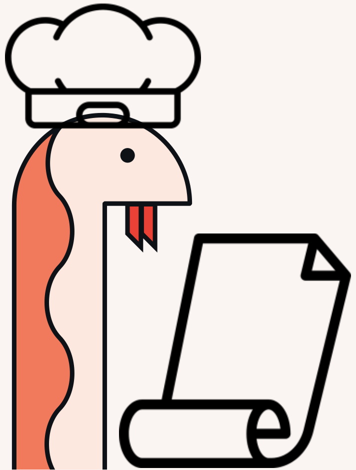
Checklist Revisited ...

- Python Ecosystem Integrated Tools
- High-Level API for computations
- Hardware Acceleration
- Production Models
- Data Handling
- Distributed Training
- Losses, Performance Metrics
- Logging, Tracking, Checkpointing
- Configuring Hyperparameters
- Using other model repositories
- Implementing different Tasks
- Reusing submodules
- Reproducibility
- Visualizing Model Predictions
- Sharing Experiment Configurations

Checklist Revisited ...

- Python Ecosystem Integrated Tools
- High-Level API for computations
- Hardware Acceleration
- Production Models
- Data Handling
- Distributed Training
- Losses, Performance Metrics
- Logging, Tracking, Checkpointing
- Configuring Hyperparameters

- Using other model repositories
- Implementing different Tasks
- Reusing submodules
- Reproducibility
- Visualizing Model Predictions
- Sharing Experiment Configurations



Quadra

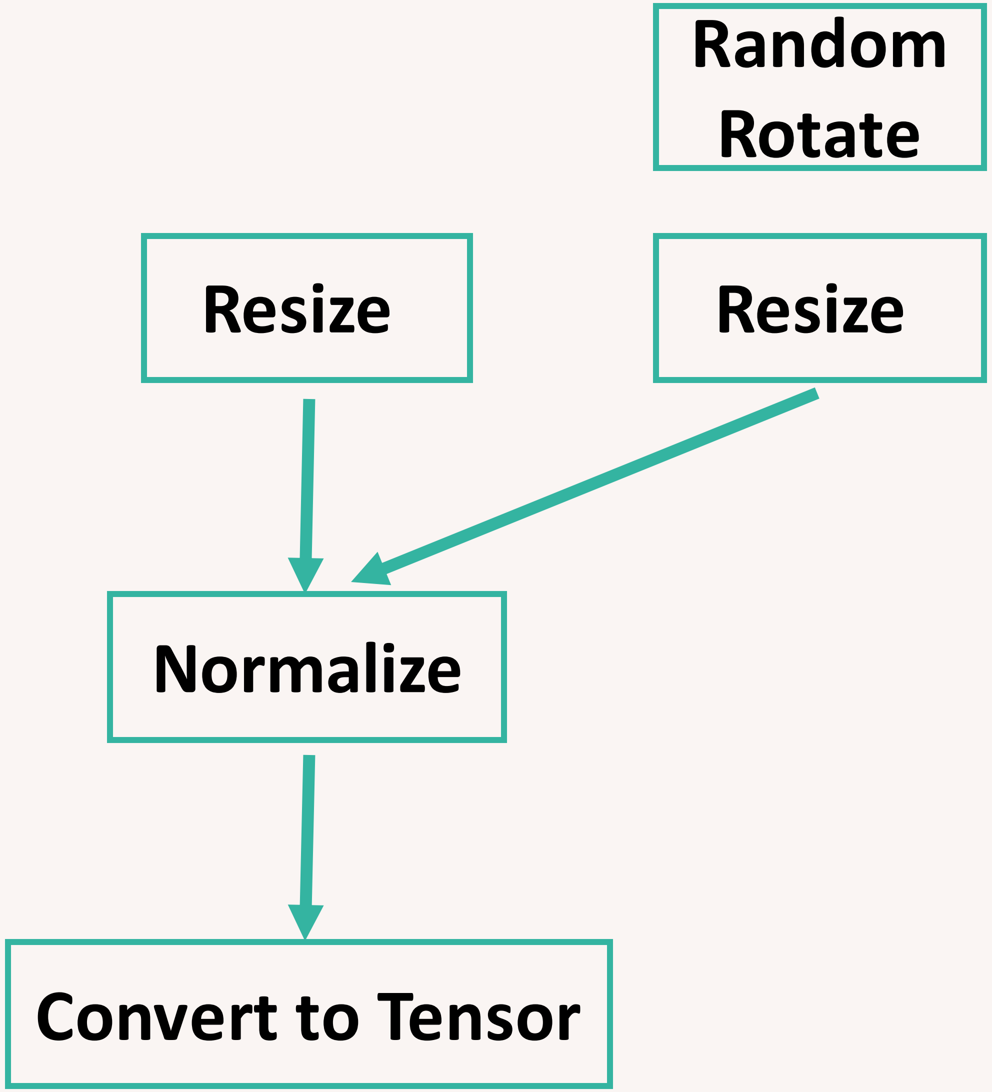
Configuration Folders

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

- **Divided into categories**
- **Contains part of the configuration**
- **Each folder can have default configuration**
- **Default configs can be extended by adding new YAML files**

Transformations

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```



Transformations

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
input_height: 256
input_width: 256
mean: [0.485, 0.456, 0.406]
std: [0.229, 0.224, 0.225]

normalize:
  _target_: albumentations.Compose
  transforms:
    - _target_: albumentations.Normalize
      mean: ${transforms.mean}
      std: ${transforms.std}
      always_apply: True
    - _target_: albumentations.pytorch.ToTensorV2
      always_apply: True

resize:
  _target_: albumentations.Resize
  height: ${transforms.input_height}
  width: ${transforms.input_width}
  interpolation: 2
  always_apply: True
standard_transform:
  _target_: albumentations.Compose
  transforms:
    - ${transforms.resize}
    - ${transforms.normalize}

train_transform: ${transforms.standard_transform}
val_transform: ${transforms.standard_transform}
test_transform: ${transforms.standard_transform}
```

Datamodules

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
_target_: quadra.datamodules.classification.ClassificationDataModule
data_path: ???
exclude_filter: ["unwanted_class"]
seed: ${core.seed}
num_workers: 8
batch_size: 16
val_size: 0.2
train_transform: ${transforms.train_transform}
test_transform: ${transforms.test_transform}
val_transform: ${transforms.val_transform}
dataset:
  _target_: hydra.utils.get_method
  path: quadra.datasets.classification.ClassificationDataset
```

Backbone Modules

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
model:
  _target_: quadra.models.classification.TorchHubNetworkBuilder
  repo_or_dir: facebookresearch/dino:main
  model_name: dino_vitb8
  pretrained: true
  freeze: false
metadata:
  input_size: 224
  output_dim: 768
  patch_size: 8
  nb_heads: 12
```

```yaml
model:
  _target_: quadra.models.classification.TimmNetworkBuilder
  model_name: resnet50
  pretrained: true
  freeze: false
metadata:
  input_size: 224
  output_dim: 2048
```

Models

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
model: ${backbone.model}
num_classes: ???
pre_classifier: null
classifier:
  _target_: torch.nn.Linear
  in_features: ${backbone.metadata.output_dim}
  out_features: ${model.num_classes}
module:
  _target_: quadra.modules.classification.ClassificationModule
  lr_scheduler_interval: "epoch"
  criterion: ${loss}
  gradcam: true
```

Callbacks

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
model_checkpoint:
  _target_: pytorch_lightning.callbacks.ModelCheckpoint
  monitor: "val_loss" # name of the logged metric which determines when model is improving
  mode: "min" # can be "max" or "min"
  save_top_k: 1 # save k best models (determined by above metric)
  save_last: True # additionaly always save model from last epoch
  verbose: False
  dirpath: "checkpoints/"
  filename: "epoch_{epoch:03d}"
  auto_insert_metric_name: False

log_gradients:
  _target_: quadra.callbacks.mlflow.LogGradients
  norm: 2
lr_monitor:
  _target_: pytorch_lightning.callbacks.LearningRateMonitor
  logging_interval: "epoch"
progress_bar:
  _target_: pytorch_lightning.callbacks.progress.TQDMProgressBar
lightning_trainer_setup:
  _target_: quadra.callbacks.lightning.LightningTrainerBaseSetup
  log_every_n_steps: 1
```

Hydra Settings

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
hydra:
  run:
    dir: logs/runs/${core.name}/${now:%Y-%m-%d_%H-%M-%S}
  sweep:
    dir: logs/multiruns/${core.name}/${now:%Y-%m-%d_%H-%M-%S}
    subdir: ${multirun_subdir_beautify:${hydra.job.override_dirname}}
  job:
    chdir: true
```

```python
def multirun_subdir_beautify(subdir: str) -> str:
    hydra_cfg = HydraConfig.get()
    if hydra_cfg.mode is None or hydra_cfg.mode.name == "RUN":
        return subdir
    subdir_list = subdir.replace("/", "|").split(",")
    subdir = ",".join([x.split("=")[1].replace(" ", "") for x in subdir_list])
    return subdir
OmegaConf.register_new_resolver("multirun_subdir_beautify", multirun_subdir_beautify)
```

```
quadra experiment=anomaly/padim trainer.batch_size=32,64 --multirun

logs/multiruns/myexperiment/2023-05-28_12-40-00/anomaly|padim,32
logs/multiruns/myexperiment/2023-05-28_12-40-00/anomaly|padim,64
```

Tasks

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

Available Tasks:

- Classification
- Segmentation
- Self-supervised Learning
- Anomaly Detection

```yaml
_target_: quadra.tasks.Classification
export_type: [torchscript]
lr_multiplier: null
output:
  example: false
report: false
run_test: true
```

Tasks

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

Task :

- Prepares modules
- Trains
- Tests
- Exports the model
- Evaluates the exported model
- Create performance report
- Visualize results

Experiments

```
configs/
├── backbone
├── callbacks
├── core
├── datamodule
├── experiment
├── hydra
├── logger
├── loss
├── model
├── optimizer
├── scheduler
├── task
├── trainer
└── transforms
```

```yaml
# @package _global_
defaults:
  - override /backbone: resnet18
  - override /datamodule: generic/imagenette/classification/base
  - override /loss: cross_entropy
  - override /model: classification
  - override /optimizer: adam
  - override /task: classification
  - override /scheduler: rop
  - override /transforms: default_resize

core:
  tag: "run"
  name: classification_imagenette_${trainer.max_epochs}

trainer:
  max_epochs: 20

model:
  num_classes: 10

logger:
  mlflow:
    experiment_name: imagenette_classification
    run_name: ${core.name}
```

Running Experiments

Classification

```
quadra experiment=generic/imagenette/classification/default
```

Running Experiments

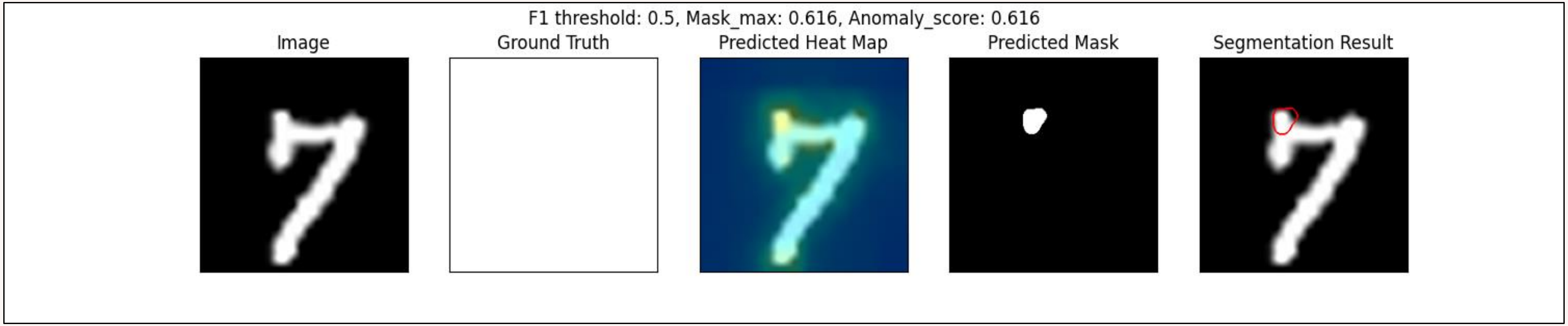
Segmentation

```
quadra experiment=generic/oxford_pet/segmentation/smp
```

Running Experiments

Anomaly Detection

```
quadra experiment=generic/mnist/anomaly/padim
```



Running Experiments

Classification

```
quadra experiment=generic/imagenette/classification/default
```

Segmentation

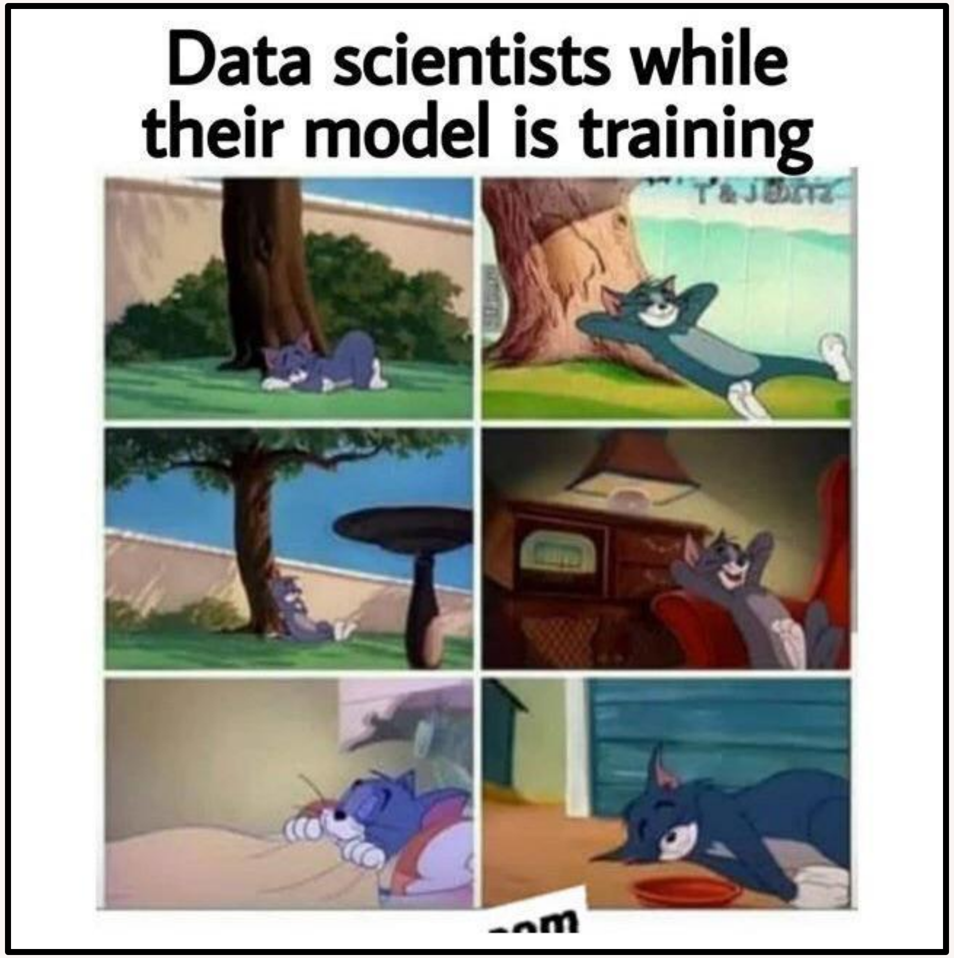
```
quadra experiment=generic/oxford_pet/segmentation/smp
```

SSL Self-supervised Learning

```
quadra experiment=generic/imagenette/ssl/byol
```

Anomaly Detection

```
quadra experiment=generic/mnist/anomaly/padim
```



@reddit/ProgrammerHumor

Reproducibility

```
logs/.../experiment_name/
├── checkpoints
│   └── ...
├── config_resolved.yaml
├── config_tree.txt
├── data
│   └── datamodule.pkl
├── deployment_model
│   └── ...
├── main.log
├── .hydra
│   ├── config.yaml
│   └── overrides.yaml
└── task_specific_outputs...
```

Experiment Folder :

- All checkpoints saved during training
- Configuration file to reload experiment
- Datamodule metadata such as splits
- Git Commit Hash
- Exported model for deployment
- Hydra settings saved
- Images, results, metrics, all data

Quadra

- **Fast Experimenting**
- **Configuration sharing among the developers**
- **Self-contained tracking, saving system with open-source tools**
- **Code reusing for different tasks**
- **Standardized results and export scheme for easy comparison**
- **Future:**
  - Hyperparameter Optimization
  - More tasks, models, configurations
  - Cloud integration

https://github.com/orobix/quadra

Shout-Out

OROBIX

- Lorenzo Mammana
- Alessandro Polidori
- Federico Belotti
- Silvia Bianchetti
- Lisa Lozza
- Luca Antiga

TALK

ChatGPT and Minecraft's diamonds playing agents: Reinforcement Learning in a nutshell with sheeprl

Intermediate, English

Federico Belotti

SheepRL - Distributed Reinforcement Learning accelerated by Lightning Fabric

https://github.com/Eclectic-Sheep/sheeprl

Thank you!