ClearML guide

Tags clearml mlops

To Do

- [x] ~~Clean and update notes~~
- [x] ~~Better logging~~
- [x] ~~Data versioning~~
- [ ] Deployment and monitoring
- [x] ~~Pipelines~~
- [ ] CI/CD

To learn

Setup

```
pip install clearml
```

Set up connection with the server.

```
clearm-init
```

Go to `app.clear.ml` and head over to `settings/workspace`.

Click on *create new credentials*. Copy these and paste in the terminal.

ClearML Agent

Used to execute tasks on remote servers. Can also schedule tasks etc.

Reference link → https://clear.ml/docs/latest/docs/clearml_agent/

```
pip install clearml-agent
clearml-agent init
```

- Agent can run in *pip, conda, poetry and docker mode.*

To run in `docker` mode

```
clearml-agent daemon --docker
```

After this, the `agent` will start listening for tasks on the default `queue`.

Initialize the agent before putting the task in queue.

- We can put multiple tasks in the queue and agent will run them one after the other.
- Can also pass desired image to the agent

```
clearml-agent daemon --docker nvidia/cuda:11.7.1-cudnn8-runtime-ubuntu22.04
```

Notes

- Import and create task.

```
import tensorflow as tf
from clearml import Task

# Task.add_requirements("./requirements.txt")

task = Task.init(project_name="mnist_example", task_name="local_training")

parameters = {"lr": 0.001}

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ]
)

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

task.connect(parameters)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=parameters["lr"]),
    loss=loss_fn,
    metrics=["accuracy"],
)

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test, verbose=2)
```

- Command-line arguments are captured automatically.
- Logs, source codes, hyper-parameters, plots and images are all captured automatically.
- For `dict` and python `objects` we can use `Task.connect()`.

  ```
  parameters = {"change_me": "value", "dropout": 0.5}

  task.connect(parameters)
  ```

- We can clone the task, change hyper-parameters and execute the task remotely.
- Experiment and data tracking
  - *Dataset version control*
  - Compare multiple experiment runs, keep track of experiment configurations etc.
- **Artifacts** like model weights, processed features, files are also stored/captured
  - Model artifacts can exist as standalone entities
    - We don't need to find the original task and then try to get the attached artifact
  - We can pull the model by its `id` or `tag`
- **Scalars**
  - Loss, accuracy etc. are stored and plotted
- Compare multiple experiment using UI
- Can also add *requirements*

```
Task.add_requirements("./requirements.txt")
```

- Reporting scalars

```
Logger.current_logger().report_scalar(
        "train", "accuracy", value=train_acc, iteration=epoch
    )
```

- We can create project and run multiple experiments inside that project

---

Experiment Tracking

Examples

https://github.com/pytholic/ClearML/blob/main/experimentation/pytorch/pytorch_test.py

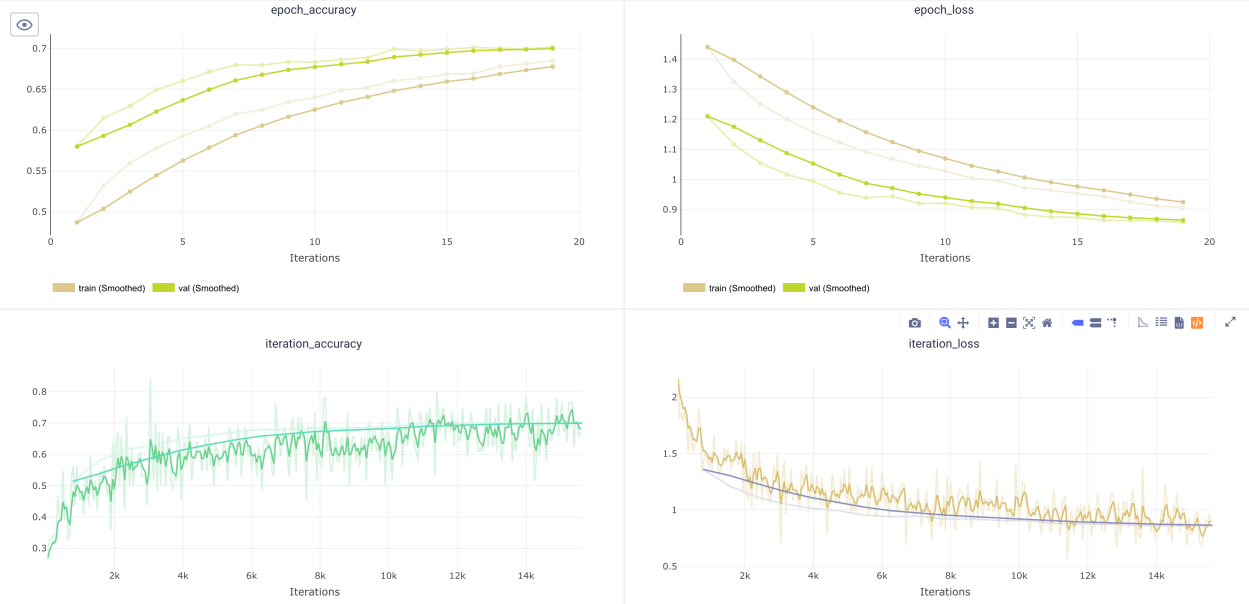
https://github.com/pytholic/ClearML/blob/main/experimentation/pytorch_lightning/cifar10_tensorboard_logger.py

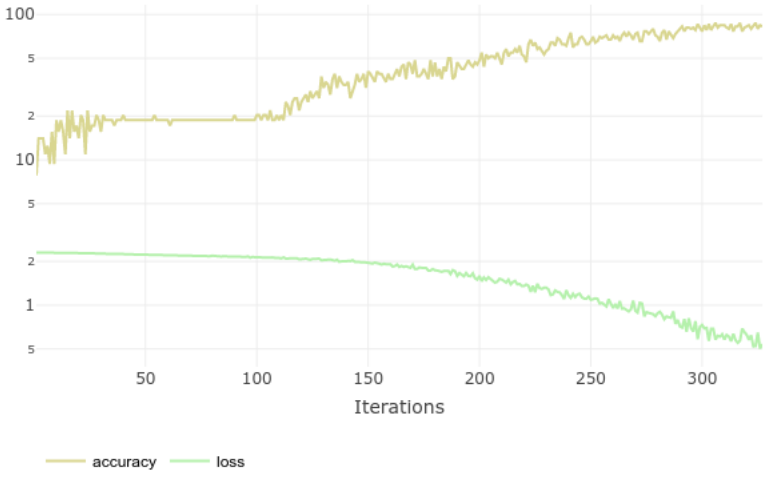
Testing on CIFAR-10 dataset. We can log loss and accuracy.

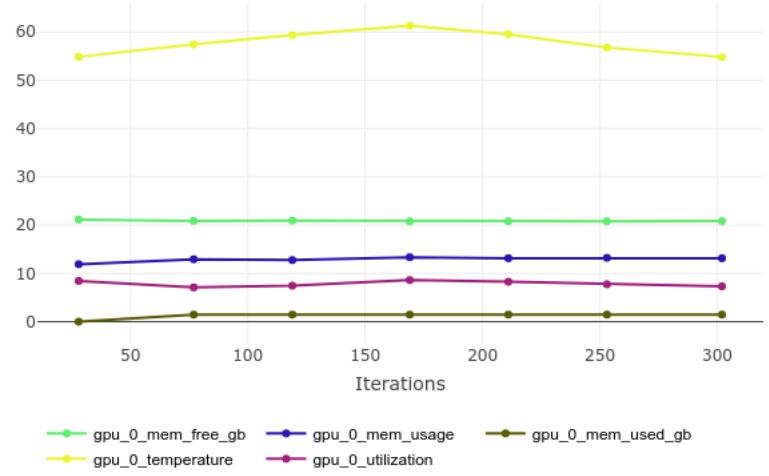
In addition, we can also get machine and gpu usage.

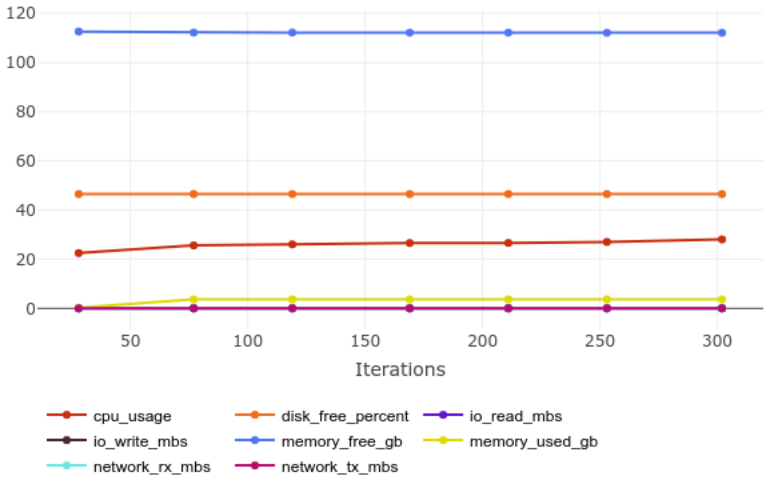
There are various kinds of loggers available especially in pytorch-lightning. I am using tensorboard below for better customization.

Ref → https://pytorch-lightning.readthedocs.io/en/stable/extensions/logging.html


epoch_accuracy
0.7
0.65
0.6
0.55
0.5
0
5
10
15
20
Iterations
train (Smoothed)
val (Smoothed)
epoch_loss
1.4
1.3
1.2
1.1
1
0.9
0
5
10
15
20
Iterations
train (Smoothed)
val (Smoothed)
iteration_accuracy
0.8
0.7
0.6
0.5
0.4
0.3
2k
4k
6k
8k
10k
12k
14k
Iterations
iteration_loss
2
1.5
1
0.5
2k
4k
6k
8k
10k
12k
14k
Iterations


100
5
2
10
5
2
1
5
50
100
150
200
250
300
Iterations
accuracy
loss





Links

Tutorial

https://www.youtube.com/watch?v=kyOfwVg05EM&list=PLMdIlCuMqSTnoC45ME5_JnsJX0zWqDdlO&index=4

Hyperparameter Optimization

One way can manually writing the configurations

```
# Define the hyperparameters for each task
hyperparameters = [
    {'learning_rate': 0.01, 'batch_size': 32, 'epochs': 5},
    {'learning_rate': 0.001, 'batch_size': 64, 'epochs': 10},
    {'learning_rate': 0.0001, 'batch_size': 128, 'epochs': 20}
]

# Train the CNN with each set of hyperparameters
for hyperparameters in Hyperparameters:
    learning_rate = hyperparameters['learning_rate']
    batch_size = hyperparameters['batch_size']
    epochs = hyperparameters['epochs']
```

There are also optimizers available in clearml.

Example link

https://github.com/pytholic/ClearML/blob/main/experimentation/hyperparameter/hyperparameter_optimization_optuna.py

Ran bunch of experiments with different configurations.

```
hyper_parameters=[
        UniformParameterRange("Args/lr", min_value=0.01, max_value=0.3, step_size=0.05),
        DiscreteParameterRange("Args/net", values=["Net1", "Net2"]),
        DiscreteParameterRange("Args/epochs", values=[10, 30, 50]),
    ]
```

We get a nice comparison in the end with descriptive titles for each experiment. All the models artifacts are saved as well for each run.

To put a time limit or task limit on the hyper-parameter optimization process.

Can use following parameter

```
optimizer.set_time_limit(in_minutes=2.0)
```

We can also set time for each job. This will limit the execution time of a single experiment, in minutes.

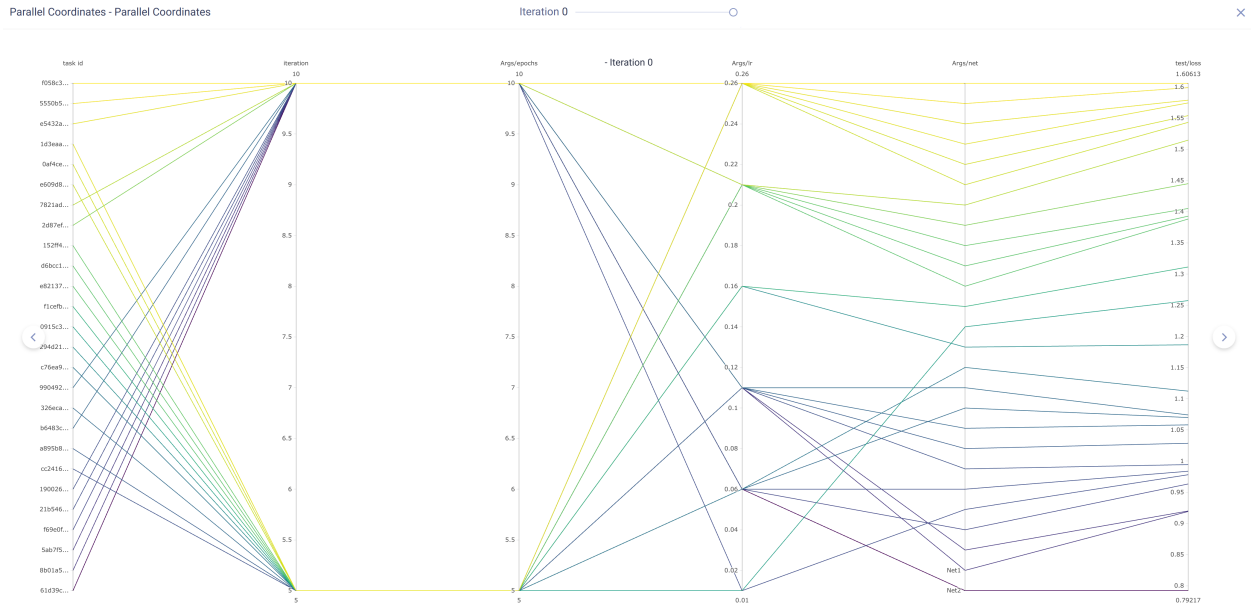
```
time_limit_per_job=10
```

Experiments that are in progress when the optimizer stop will continue to run. However, artifacts might not be stored since the main process stopped.

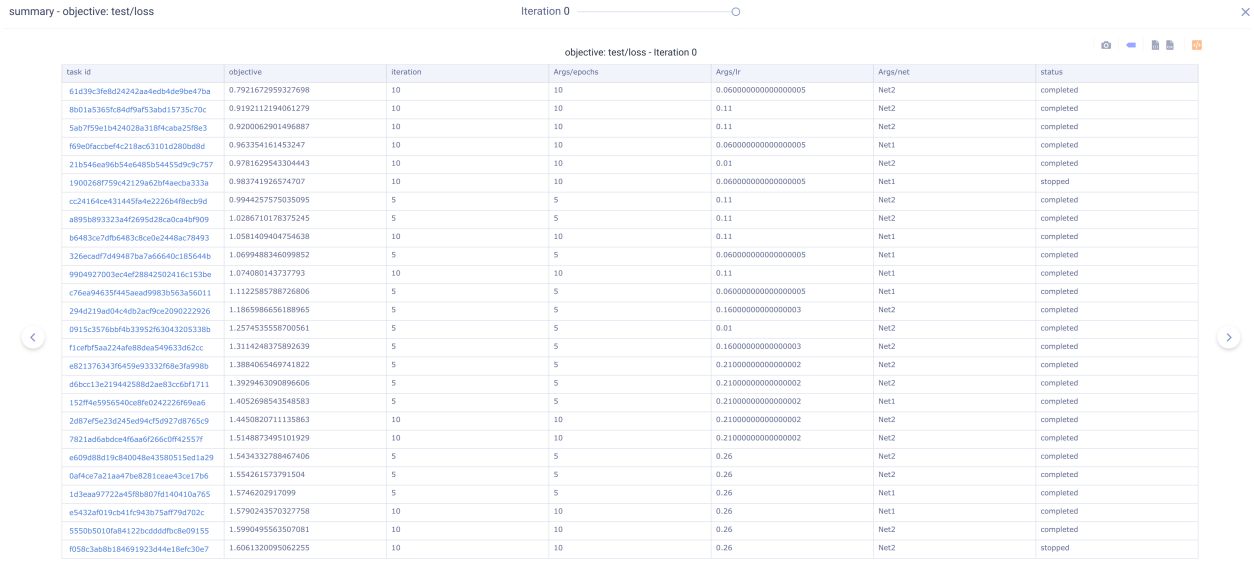
Plots

Can see the final objective for all the jobs

Different task configurations and final metrics



Objective leaderboards



task id	objective	iteration	Args/epochs	Args/lr	Args/net	status
61d39c3fe8d24242aa4edb4de9be47ba	0.7921672959327698	10	10	0.060000000000000005	Net2	completed
8b01a5365fc84df9af53abd15735c70c	0.9192112194061279	10	10	0.11	Net2	completed
5ab7f59e1b424028a318f4caba25f8e3	0.9200062901496887	10	10	0.11	Net2	completed
f69e0faccbef4c218ac63101d280bd8d	0.963354161453247	10	10	0.060000000000000005	Net1	completed
21b546ea96b54e6485b54455d9c9c757	0.9781629543304443	10	10	0.01	Net2	completed
1900268f759c42129a62bf4aecba333a	0.983741926574707	10	10	0.060000000000000005	Net1	stopped
cc24164ce431445fa4e2226b4f8ecb9d	0.9944257575035095	5	5	0.11	Net2	completed
a895b893323a4f2695d28ca0ca4bf909	1.0286710178375245	5	5	0.11	Net2	completed
b6483ce7dfb6483c8ce0e2448ac78493	1.0581409404754638	10	10	0.11	Net1	completed
326ecadf7d49487ba7a66640c185644b	1.0699488346099852	5	5	0.060000000000000005	Net1	completed
9904927003ec4ef28842502416c153be	1.074080143737793	10	10	0.11	Net1	completed
c76ea94635f445aead9983b563a56011	1.1122585788726806	5	5	0.060000000000000005	Net1	completed
294d219ad04c4db2acf9ce2090222926	1.1865986656188965	5	5	0.16000000000000003	Net2	completed
0915c3576bbf4b33952f6304320538bb	1.2574535558700561	5	5	0.01	Net2	completed
f1cefbf5aa224afe88dea549633d62cc	1.3114248375892639	5	5	0.16000000000000003	Net2	completed
e82137634 3f6459e93332f68e3fa998b	1.3884065469741822	5	5	0.21000000000000002	Net2	completed
d6bcc13e219442588d2ae83cc6bf1711	1.3929463090896606	5	5	0.21000000000000002	Net2	completed
152ff4e5956540ce8fe0242226f69ea6	1.4052698543548583	5	5	0.21000000000000002	Net1	completed
2d87ef5e23d245ed94cf5d927d8765c9	1.4450820711135863	10	10	0.21000000000000002	Net2	completed
7821ad6abdce4f6aa6f266c0ff42557f	1.5148873495101929	10	10	0.21000000000000002	Net2	completed
e609d88d19c840048e43580515ed1a29	1.5434332788467406	5	5	0.26	Net2	completed
0af4ce7a21aa47be8281ceae43ce17b6	1.554261573791504	5	5	0.26	Net2	completed
1d3eaa97722a45f8b807fd140410a765	1.5746202917099	5	5	0.26	Net1	completed
e5432af019cb41fc943b75aff79d702c	1.5790243570327758	10	10	0.26	Net1	completed
5550b5010fa84122bcdddfbc8e09155	1.5990495563507081	10	10	0.26	Net2	completed
f058c3ab8b18469123d44e18efc30e7	1.6061320095062255	10	10	0.26	Net2	stopped

Top experiments

We can also return to `top_k` experiments which gave best performance.

```
# Return a list of top performing experiments
top_exp = optimizer.get_top_experiments(top_k=3)
print([t.id for t in top_exp])  # print just the id
```

```
# Return a dict of top performing experiments with more details
top_exp = optimizer.get_top_experiments_details(top_k=3)
```

```
[{'task_id': '3d5c98024baa47bc95e065715ca6724b', 'hyper_parameters':
{'Args/net': 'Net2', 'Args/epochs': '10', 'Args/lr': '0.11'},
'metrics': {'test/loss': {'metric': 'test', 'variant': 'loss',
'value': 0.9133659139633179, 'min_value': 0.9133659139633179,
'min_value_iteration': 10, 'max_value': 1.9955348001480102, 'max_value_iteration': 1}}},
{'task_id': '62328586c8034ae6969151e205c0a90b',
'hyper_parameters': {'Args/net': 'Net2', 'Args/epochs': '5', 'Args/lr': '0.11'},
'metrics': {'test/loss': {'metric': 'test', 'variant': 'loss',
'value': 1.0147933155059814, 'min_value': 1.0147933155059814,
'min_value_iteration': 5, 'max_value': 1.697532457923889,
'max_value_iteration': 1}}},
{'task_id': 'db87c34586c945ffb33cbcef21a66a82',
'hyper_parameters': {'Args/net': 'Net1', 'Args/epochs': '10', 'Args/lr': '0.11'},
'metrics': {'test/loss': {'metric': 'test', 'variant': 'loss',
'value': 1.1441912442207336, 'min_value': 1.1441912442207336,
'min_value_iteration': 5, 'max_value': 1.500229824066162,
'max_value_iteration': 1}}}]
```

I had to place it between `wait` and `stop` for it to work.

```
# wait until optimization completed or timed-out
optimizer.wait()

# top_exp = optimizer.get_top_experiments(top_k=3)
# print([t.id for t in top_exp])

top_exp = optimizer.get_top_experiments_details(top_k=3)

# make sure we stop all jobs
optimizer.stop()
```

Job callbacks

We can also add `job_complete_callback` to get console message whenever a job is completed along with the experiment details.

```
Job completed! 3d5c98024baa47bc95e065715ca6724b 0.9133659139633179 10 {'status': 'in_progress', 'Args/lr': 0.11, 'Args/net': 'Net2', 'Args/
Top performance experiment id: 3d5c98024baa47bc95e065715ca6724b
```

Optuna Optimizer

Need to have optuna installed.

```
pip install optuna
```

Seems like there are some required *parameters.*

```
TypeError: OptimizerOptuna.__init__() missing 2 required positional arguments: 'max_iteration_per_job' and 'total_max_jobs'
```

Parameters for `OptimzierOptuna` are slightly different from normal `HyperparameterOptimizer`.

We just set the class and parameters and run like the last example.

```
optimizer_class=OptimizerOptuna
```

💡 One useful option `total_max_jobs` is also available with `OptimzierOptuna`. It was missing in normal `RandomSearch` optimizer.

Links

Documentation

https://clear.ml/docs/latest/docs/fundamentals/hpo

Tutorial

https://www.youtube.com/watch?v=aY1ImY4OnKk&list=PLMdIlCuMqSTnoC45ME5_JnsJX0zWqDdlO&index=6

ClearML Data

Use ClearML Data to create, manage, and version your datasets.

Accessibility → Data can be accessed from any machine

Versioning → Linking which data set version was used in which task

This helps in reproducibility.

Creating dataset with CLI

Steps

Initialize the dataset

First we need to `create` an initial dataset version.

We can do it with CLI.

```
clearml-data create --project experimentation/clearml-data --name animal-data
```

Add files

This will initialize the dataset, but it will be empty. We can then use `add` command to add files.

```
clearml-data add --files ./images
```

```
clearml-data - Dataset Management & Versioning CLI
Adding files/folder/links to dataset id 399a206cdda04949b262f11a553552a5
Generating SHA2 hash for 90 files
100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████
Hash generation completed
90 files added
```

This will recursively add files to the dataset.

Close

Now we need to tell the server that we are done here.

```
clearml-data - Dataset Management & Versioning CLI
Finalizing dataset id 399a206cdda04949b262f11a553552a5
Pending uploads, starting dataset upload to https://files.clear.ml
Uploading dataset changes (90 files compressed to 3.96 MiB) to https://files.clear.ml
File compression and upload completed: total size 3.96 MiB, 1 chunk(s) stored (average size 3.96 MiB)
Dataset closed and finalized
```

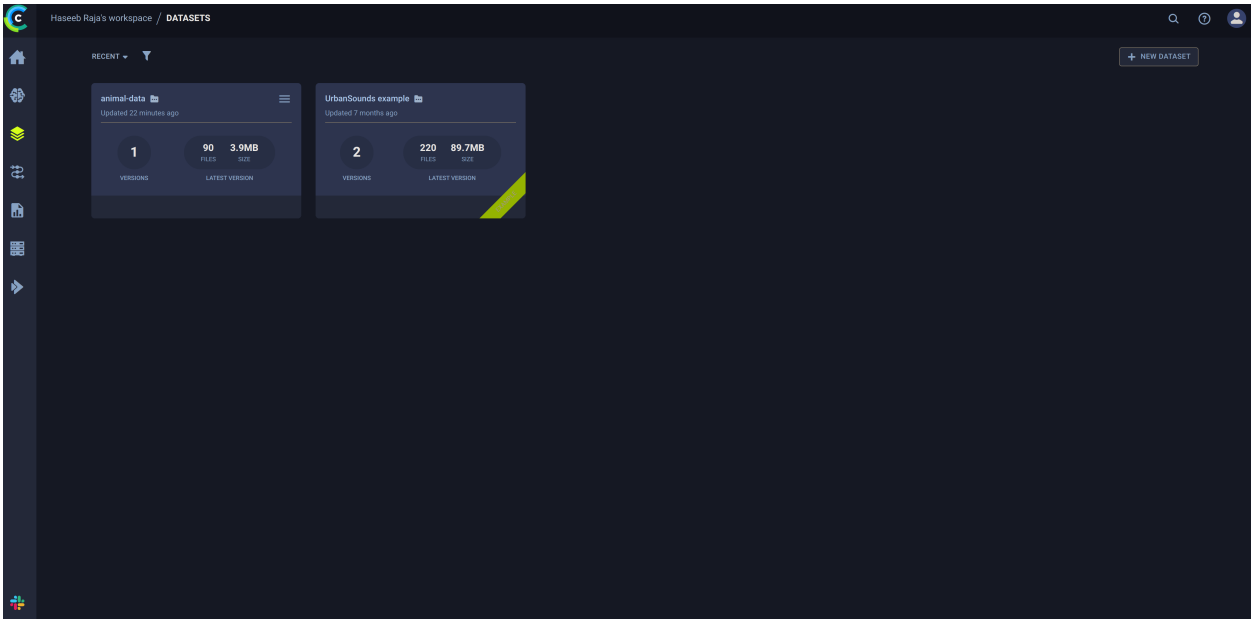
It will *upload* the files and change the dataset status to *finalized*. This will lock this version of the dataset.

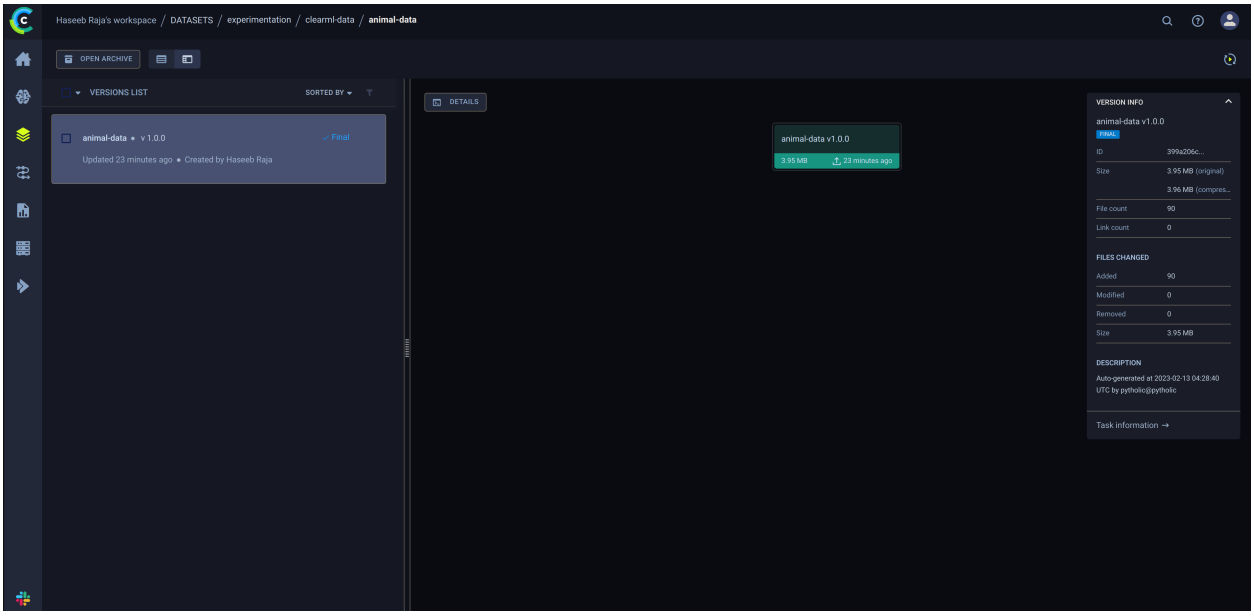
Creating dataset with Python SDK

```
import os
from pathlib import Path

from clearml import Dataset

DATA_ROOT = "./clearml-data/sample_data/images"

# # Create dataset
ds = Dataset.create(dataset_name="animal-dataset", dataset_project="clearml-data")

# # Add files
ds.add_files(path=DATA_ROOT)

# Count the amount of samples per class
root_folder = Path(DATA_ROOT)
counts = []
folders = sorted(os.listdir(root_folder))
for folder in folders:
    counts.append([len(os.listdir(root_folder / folder))])

# Log some dataset statistics
ds.get_logger().report_histogram(
    title="Dataset Statistics",
    series="Train Test Split",
    labels=folders,
    values=counts,
)

# Finalize and upload
# ds.upload()
# ds.finalize()
ds.finalize(auto_upload=True)
```

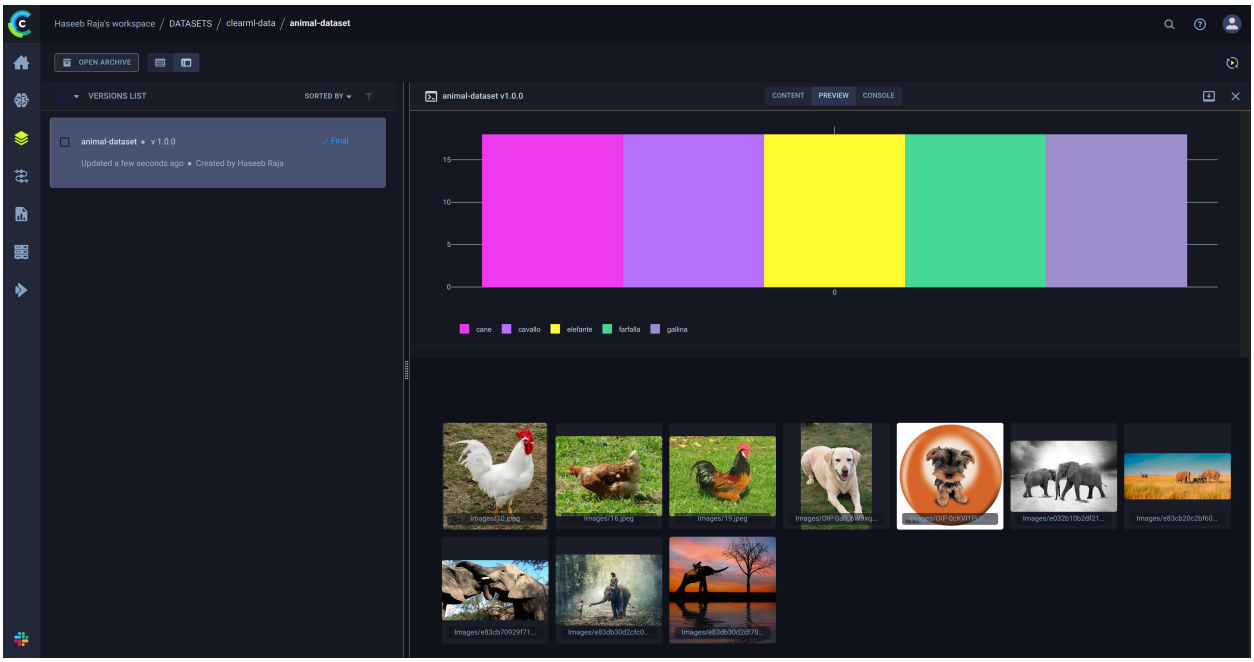
```
ClearML results page: https://app.clear.ml/projects/23aaa92afe0d4c919b74c9770620ad77/experiments/224508a1e13b4d96a1a45209a3f0c049/output/lo
ClearML dataset page: https://app.clear.ml/datasets/simple/23aaa92afe0d4c919b74c9770620ad77/experiments/224508a1e13b4d96a1a45209a3f0c049
Generating SHA2 hash for 90 files
100%|
Hash generation completed
Pending uploads, starting dataset upload to https://files.clear.ml
Uploading dataset changes (90 files compressed to 3.96 MiB) to https://files.clear.ml
File compression and upload completed: total size 3.96 MiB, 1 chunk(s) stored (average size 3.96 MiB)
```

Dataset UI





We can also see *samples* and *histogram*.



Downloading dataset

You can download this version from any machine.

CLI

```
clearml-data get --id 224508a1e13b4d96a1a45209a3f0c049

# Get in desired folder
clearml-data get --name "animal-data" --copy /clearml-data/tmp/my_dataset_2
```

```
clearml-data - Dataset Management & Versioning CLI
Download dataset id 224508a1e13b4d96a1a45209a3f0c049
Dataset local copy available for files at: /home/pytholic/.clearml/cache/storage_manager/datasets/ds_224508a1e13b4d96a1a45209a3f0c049
```

You can find if from the UI or by searching.

```
clearml-data search --name "animal-data"  # can also use tags/projects to search
```

```
clearml-data - Dataset Management & Versioning CLI
Search datasets
project          | name                          | tags               | created             | id
-----------------------------------------------------------------------------------------------------------------------------------
clearml-data     | animal-dataset                |                    | 2023-02-13 05:15:22 | 224508a1e13b4d96a1a45209a3f0c049
```

The downloaded data is cached locally. If you try to get dataset again or any other data set based on this one, it will check which files are already present and then download only the remaining ones.

SDK

The python interface has one major difference.

Getting the dataset does not mean it will be downloaded. You only capture the meta data.

> 💡 This means we don't have to download the entire dataset to make changes or add files to it.

```
from clearml import Dataset

ds = Dataset.get(dataset_name="animal-data")

# The dataset is not download but it has all the meta data
print(ds.list_files()[:5])
```

If you do wish to download the local copy of the dataset, it has to be done explicitly.

```
# Get read-only local copy of the data, now we will download
local_path_read_only = ds.get_local_copy()
print(local_path_read_only)
```

If you want to modify and make new versions

```
# Get a mutable copy of dataset to the desired folder
local_path_mutable = ds.get_mutable_local_copy(
    target_folder="/clearml-data/tmp/my_dataset"
)
```

Creating new version

We have downloaded the dataset original version.

Now we removed one folder (one category) and added a new category to create a new version of the data.

```
from clearml import Dataset

ds = Dataset.create(
    dataset_name="animal-dataset-with-mucca",
    dataset_project="clearml-data",
    parent_datasets=["224508a1e13b4d96a1a45209a3f0c049"],
)
```

We pass the id of previous original dataset version as the `parent_dataset`.

Next we modify the *metadeta* object.

```
# Now we can make changes to the dataset without downlaoding it
ds.remove_files("./gallina")
ds.add_files("./tmp/modified/mucca")
```

It will sync everything when it is finalized.

If we already have a local copy that shows how we want the dataset to look like, then in that case it is easier to use the `sync` method.

```
# If we have a local copy  that represent show we want dataset to look like
# in that case it is easier to just use the sync method
ds.sync_folder(
    local_path="./tmp/modified/"
)
```

This will compare metadata with the content of the supplied local_path.

Now when we call finalize and upload, it will only upload the files that are changed.

```
ds.finalize(auto_upload=True)
```

The CLI interface does not have the luxury of having metadata. It can only work with the sync command. However, it can bunch this whole process into a single command.

```
clearml-data sync --project clearml-data --id animal-dataset-with-mucca --parents 224508a1e13b4d96a1a45209a3f0c049 --folder /tmp/modified/
```

Now we can check the UI again.

Links

Documentation

https://clear.ml/docs/latest/docs/clearml_data/

Tutorial

https://www.youtube.com/watch?v=S2pz9jn26uI&list=PLMdIlCuMqSTnoC45ME5_JnsJX0zWqDdlO&index=9

Organizing

Before anything else, organizing code in the following format:

```
.
├── config
│   ├── args.json
│   ├── config.py
├── logs
├── main.py
├── model.py
└── utils.py
```

We have a `main.py` file and a model definition file `model.py`. A folder for `logs` and some `config` files.

Loading args

```
args_path = config.CONFIG_DIR / "args.json"
args = Namespace(**utils.load_dict(filepath=args_path))
```

This way we have everything separated and it is easy to make changes.

Remembers, since we are using external file instead of CLI arguments, we have to to `task.connect(args)` to record hyper-parameters.

Logging

Logging is the process of tracking and recording key events that occur in our applications for the purpose of inspection, debugging, etc.

```
import logging
import sys

# Create super basic logger
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Logging levels (from lowest to highest priority)
logging.debug("Used for debugging your code.")
logging.info("Informative messages from your code.")
logging.warning("Everything works but there is something to be aware of.")
logging.error("There's been a mistake with the process.")
logging.critical("There is something terribly wrong and process may terminate.")
```

```
DEBUG:root:Used for debugging your code.
INFO:root:Informative messages from your code.
WARNING:root:Everything works but there is something to be aware of.
ERROR:root:There's been a mistake with the process.
CRITICAL:root:There is something terribly wrong and process may terminate.
```

Use use RichHandler for our `console` handler to get pretty formatting for the log messages.

```
pip install rich
```

```
import logging
import logging.config
from rich.logging import RichHandler
```

For my case, I just need `logger.info()` for now.

```
from config.config import logger
...
logger.info("Initilizaing clearML task...")
    task = Task.init(
        project_name="experimentation/logging",
        task_name=f"logging-example-{datetime.now()}",
    )
```

How it looks like

```
~/pr/p/c/practice/pipelines  main !3
❯ python3 main.py
[02/22/23 15:13:33] INFO     Created a temporary directory at   instantiator.py:21
                             /tmp/tmpgb1hm80c
                    INFO     Writing                            instantiator.py:76
                             /tmp/tmpgb1hm80c/_remote_module_no
                             n_scriptable.py
ClearML Task: created new task id=1daf2eca76054348a405a063cb5b00bc
ClearML results page: https://app.clear.ml/projects/782628c316a547089a0391c07f186f
2b/experiments/1daf2eca76054348a405a063cb5b00bc/output/log
ClearML pipeline page: https://app.clear.ml/pipelines/782628c316a547089a0391c07f18
6f2b/experiments/1daf2eca76054348a405a063cb5b00bc
[02/22/23 15:13:50] INFO     No repository found, storing script scriptinfo.py:909
                             code instead
[02/22/23 15:13:53] INFO     No repository found, storing script scriptinfo.py:909
                             code instead
[02/22/23 15:13:56] INFO     No repository found, storing script scriptinfo.py:909
                             code instead
[02/22/23 15:14:08] INFO     Reading arguments...                        main.py:106
                    INFO     Preparing data...                           main.py:119
Launching step [prepare_data]
[02/22/23 15:15:28] INFO     Creating subsets...                         main.py:122
Launching step [create_subset]
[02/22/23 15:16:29] INFO     Creating model...                           main.py:130
                    INFO     Starting training...                        main.py:134
                    INFO     GPU available: True (cuda), used:      rank_zero.py:45
                             True
                    INFO     TPU available: False, using: 0 TPU     rank_zero.py:45
                             cores
                    INFO     IPU available: False, using: 0 IPUs    rank_zero.py:45
                    INFO     HPU available: False, using: 0 HPUs    rank_zero.py:45
Launching step [train]
```

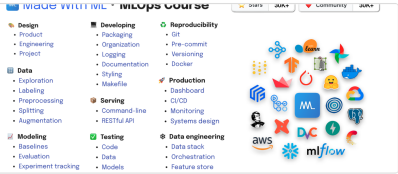
Links

Logging for ML Systems

Logging for ML Systems - Made With ML

Keep records of the important events in our application.

https://madewithml.com/courses/mlops/logging/



ClearML Pipelines

Pipelines are useful to automate and orchestrate multiple tasks. Each task is a separate `step`.

Mainly useful when each step creates an output that is an input to another set.

- Pipelines are highly scalable.
- Really useful for ETL pipelines.

In clearML we can create pipelines from `Tasks` or `Code`. In former we can create separate task for each step. In latter we can create pipelines from codebase using decorators.

I want to create a very simple pipeline

```
Load data -> process data -> feed into model
```

I am doing three things.

- Load and transform data
- Create subset (just to add an extra step)
- Feed into model

Issue

Facing an issue

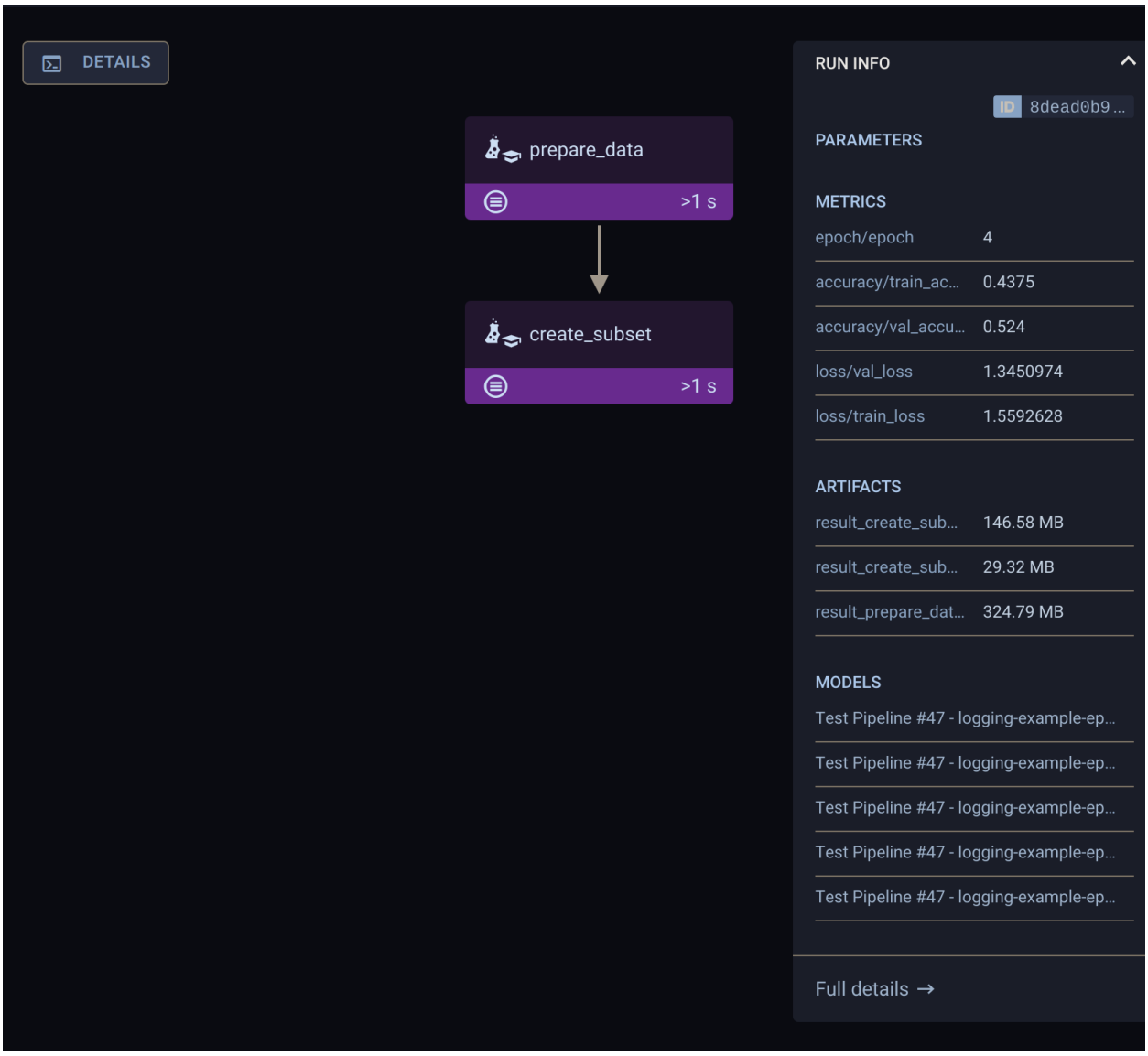
```
raise TypeError(f"`model` must be a `LightningModule`, got `{type(model).__qualname__}`")
TypeError: `model` must be a `LightningModule`, got `PosixPath`
```

Tried printing the `classifier`

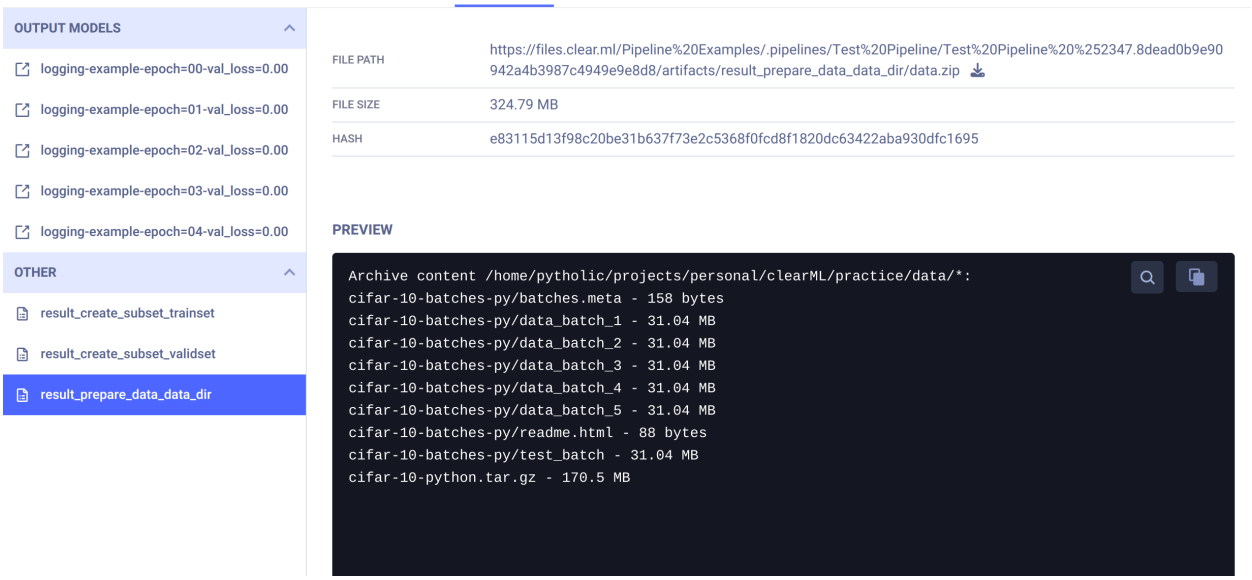
```
/home/pytholic/.clearml/cache/storage_manager/global/caa12decdd94851b2449688c2e69c588.result_train_classifier.pkl
```

It got converted to a path because of the wrapper.

Instead of creating separate train function, I ended up training it directly in the `main()` function.


DETAILS
prepare_data
>1 s
create_subset
>1 s
RUN INFO
ID 8dead0b9...
PARAMETERS
METRICS
epoch/epoch 4
accuracy/train_ac... 0.4375
accuracy/val_accu... 0.524
loss/val_loss 1.3450974
loss/train_loss 1.5592628
ARTIFACTS
result_create_sub... 146.58 MB
result_create_sub... 29.32 MB
result_prepare_dat... 324.79 MB
MODELS
Test Pipeline #47 - logging-example-ep...
Test Pipeline #47 - logging-example-ep...
Test Pipeline #47 - logging-example-ep...
Test Pipeline #47 - logging-example-ep...
Test Pipeline #47 - logging-example-ep...
Full details →

You also get all the artifacts for each step.



Tidbits

- If you are running concurrent jobs, your console might not look pretty
  - However, you can use the Experiment UI in Clear ML app to view `console` of each experiment separately

Links

Examples

clearml/examples at master · allegroai/clearml

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/allegroai/clearml/tree/master/examples

Guides

Examples | ClearML

To help learn and use ClearML, we provide example scripts that demonstrate how to use ClearML's various features.

https://clear.ml/docs/latest/docs/guides/

Documentation

What is ClearML? | ClearML

ClearML is an open source platform that automates and simplifies developing and managing machine learning solutions

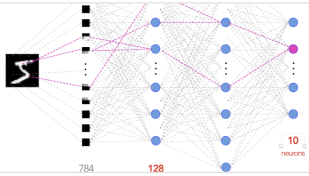
https://clear.ml/docs/latest/docs

Pytorch vs. Pytorch-Lightning

From PyTorch to PyTorch Lightning—A gentle introduction

This post walks through a side-by-side comparison of MNIST implemented using both PyTorch and PyTorch Lightning.

https://towardsdatascience.com/from-pytorch-to-pytorch-lightning-a-gentle-introduction-b371b7caaf09



ClearML SDK

Task | ClearML

class Task()

https://clear.ml/docs/latest/docs/references/sdk/task

Queries

What is the difference in task.log and Logger?

The `task.log` method is a method in the `Task` class in the ClearML library, and it is used to log scalar metric values during the training of a machine learning model. The `Logger` class, on the other hand, is a more general purpose logging tool that can be used to log metric values, image, audio and other type of data in an experiment, not limited to a single task.

In summary, you can use `task.log` to log scalar metrics during training of a model, while `Logger` can be used to log a wider range of data, not just scalar metrics. Additionally, `Logger` is typically used in the context of an experiment, while `task.log` is typically used in the context of a single task.