Deep Learning Optimisé - Jean Zay

Visualization Tools

Why use a visualization tool ?

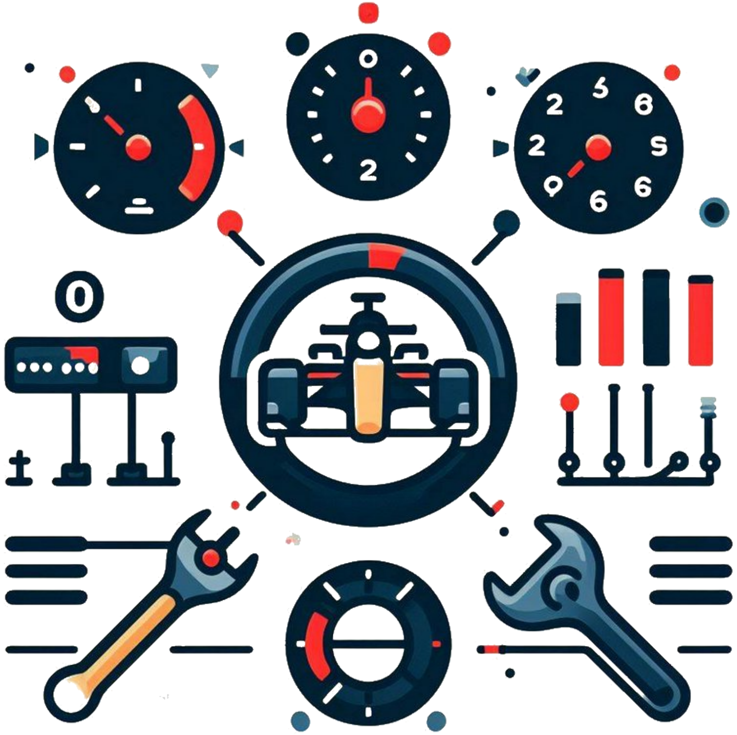
Can’t fix what you can’t see ◂

Training -> metrics ◂

Experiment -> comparaison ◂

Can’t fix what you can’t see

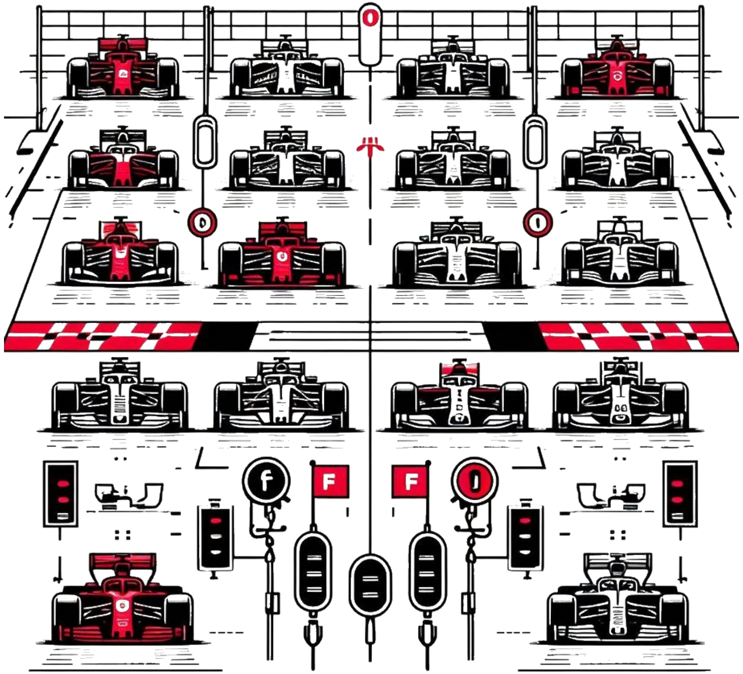
Training : metrics



Specific to learning:

- train/val loss/acc
- prototyping
- profiling & debugging

Experiment : comparaison



Not only training specific :

- hyperparameters
- dataset & source code
- hardware tracking
- multi-user / collaboration

a refined selection of tools

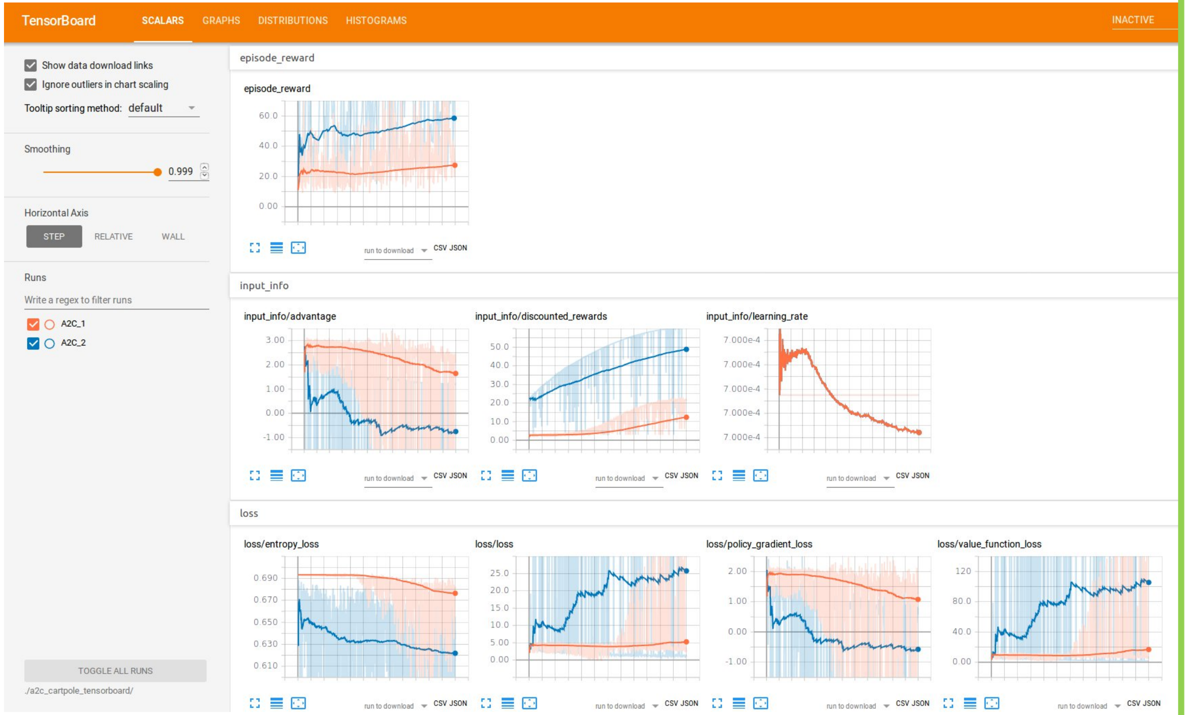
💯 TensorBoard ◀

❤ MLFlow ◀

W&B ◀

Neptune ◀

Tensorboard



☑ Show data download links

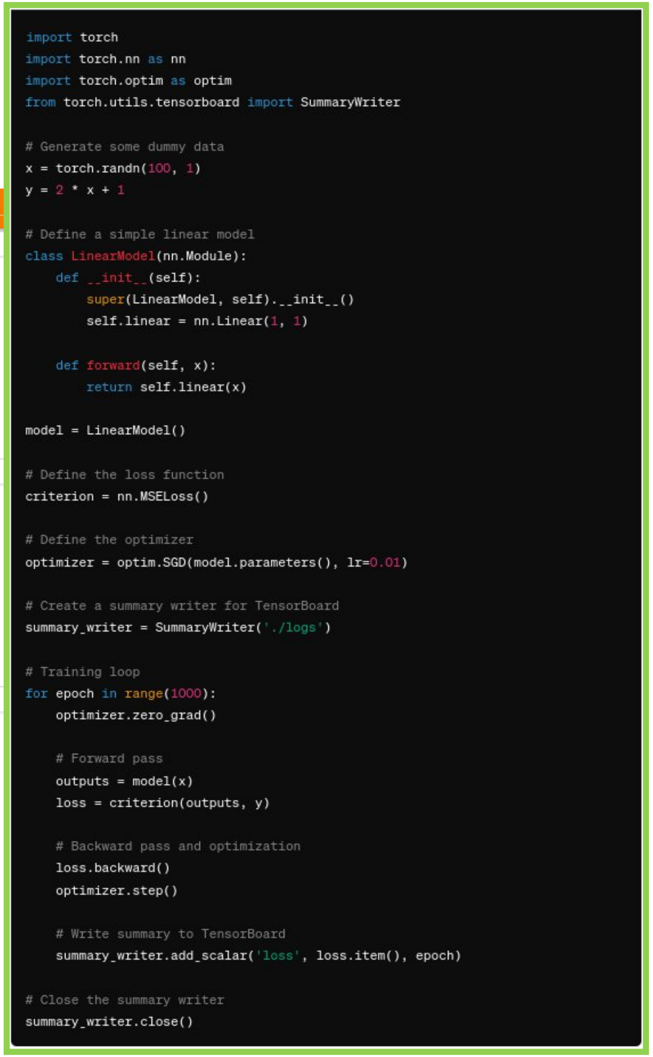
☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.999

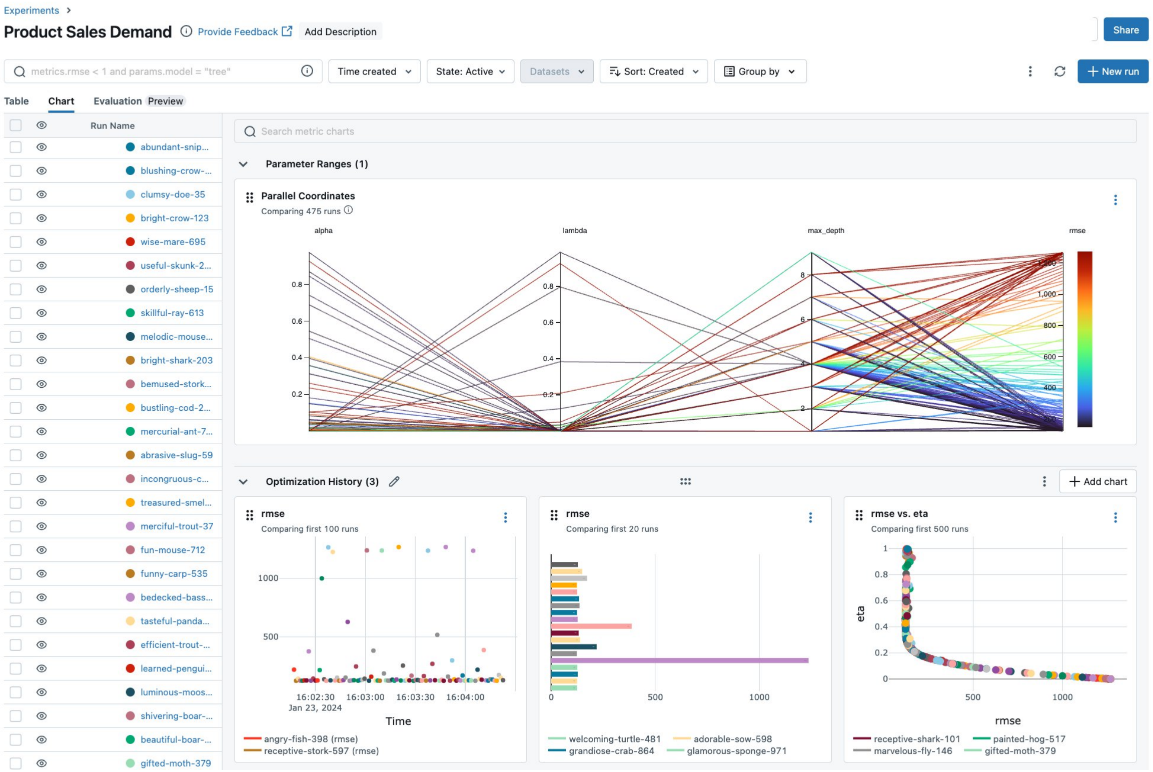
☑ A2C_1

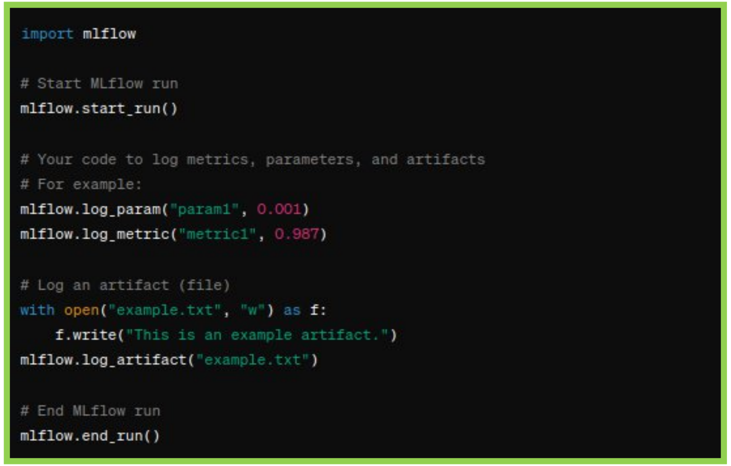
☑ A2C_2



```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.tensorboard import SummaryWriter

# Generate some dummy data
x = torch.randn(100, 1)
y = 2 * x + 1

# Define a simple linear model
class LinearModel(nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        return self.linear(x)

model = LinearModel()

# Define the loss function
criterion = nn.MSELoss()

# Define the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Create a summary writer for TensorBoard
summary_writer = SummaryWriter('./logs')

# Training loop
for epoch in range(1000):
    optimizer.zero_grad()

    # Forward pass
    outputs = model(x)
    loss = criterion(outputs, y)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()

    # Write summary to TensorBoard
    summary_writer.add_scalar('loss', loss.item(), epoch)

# Close the summary writer
summary_writer.close()
```

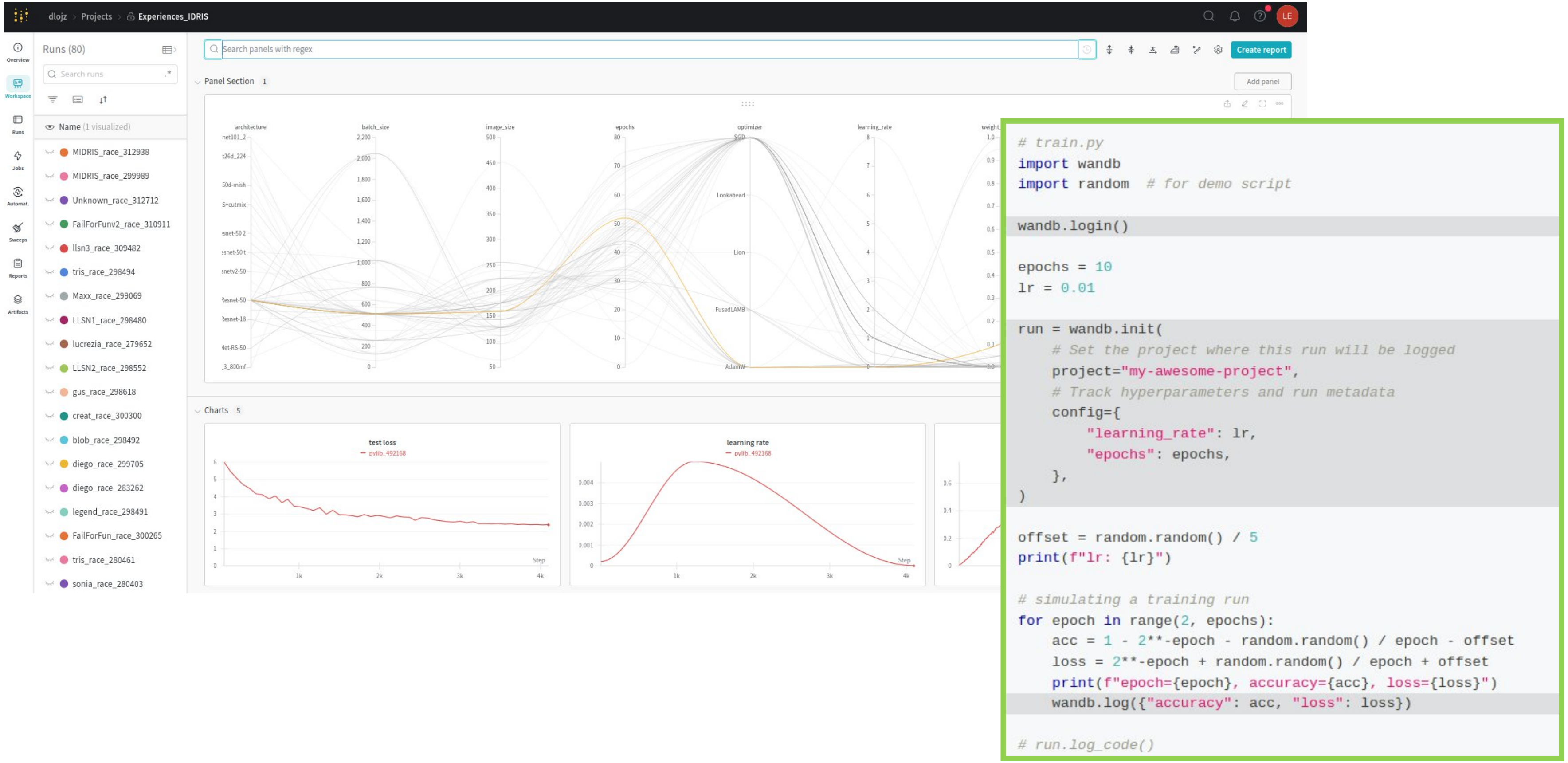
MLFlow





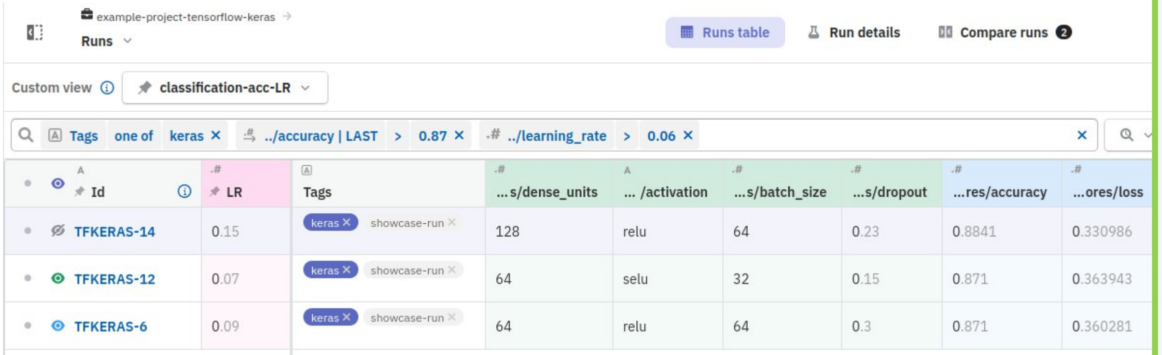
```
import mlflow

# Start MLflow run
mlflow.start_run()

# Your code to log metrics, parameters, and artifacts
# For example:
mlflow.log_param("param1", 0.001)
mlflow.log_metric("metric1", 0.987)

# Log an artifact (file)
with open("example.txt", "w") as f:
    f.write("This is an example artifact.")
mlflow.log_artifact("example.txt")

# End MLflow run
mlflow.end_run()
```

W&B



```python
# train.py
import wandb
import random  # for demo script

wandb.login()

epochs = 10
lr = 0.01

run = wandb.init(
    # Set the project where this run will be logged
    project="my-awesome-project",
    # Track hyperparameters and run metadata
    config={
        "learning_rate": lr,
        "epochs": epochs,
    },
)

offset = random.random() / 5
print(f"lr: {lr}")

# simulating a training run
for epoch in range(2, epochs):
    acc = 1 - 2**-epoch - random.random() / epoch - offset
    loss = 2**-epoch + random.random() / epoch + offset
    print(f"epoch={epoch}, accuracy={acc}, loss={loss}")
    wandb.log({"accuracy": acc, "loss": loss})

# run.log_code()
```

Neptune



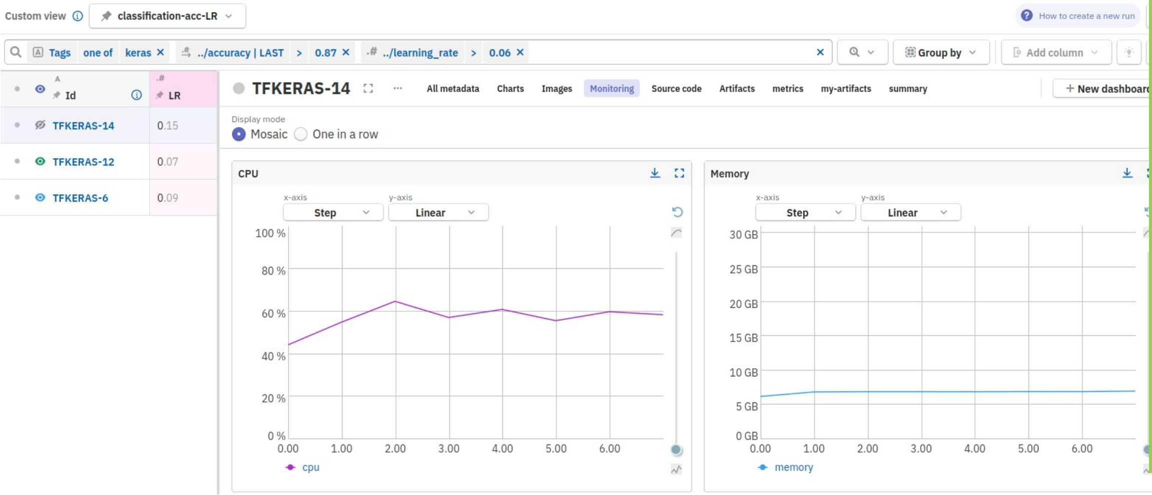
example-project-tensorflow-keras →

Runs

Custom view ⓘ classification-acc-LR

Tags one of keras × ../accuracy | LAST > 0.87 × ../learning_rate > 0.06 ×

Id	LR	Tags	...s/dense_units	... /activation	...s/batch_size	...s/dropout	...res/accuracy	...ores/loss
TFKERAS-14	0.15	keras × showcase-run ×	128	relu	64	0.23	0.8841	0.330986
TFKERAS-12	0.07	keras × showcase-run ×	64	selu	32	0.15	0.871	0.363943
TFKERAS-6	0.09	keras × showcase-run ×	64	relu	64	0.3	0.871	0.360281



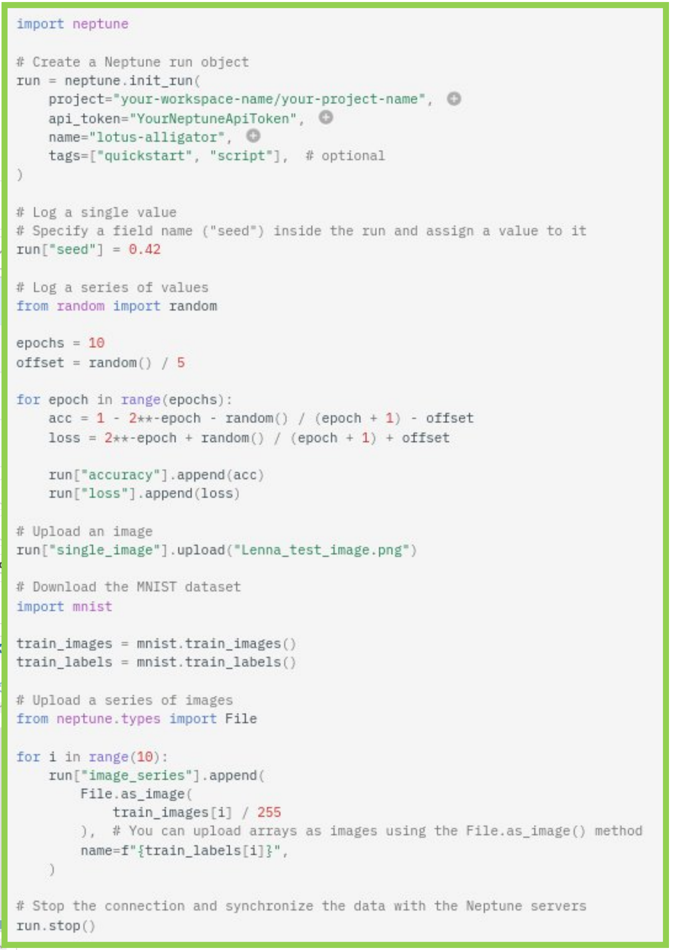
Custom view ⓘ classification-acc-LR

Tags one of keras × ../accuracy | LAST > 0.87 × ../learning_rate > 0.06 ×

Id	LR
TFKERAS-14	0.15
TFKERAS-12	0.07
TFKERAS-6	0.09

TFKERAS-14

Display mode
● Mosaic ○ One in a row



```python
import neptune

# Create a Neptune run object
run = neptune.init_run(
    project="your-workspace-name/your-project-name",
    api_token="YourNeptuneApiToken",
    name="lotus-alligator",
    tags=["quickstart", "script"],  # optional
)

# Log a single value
# Specify a field name ("seed") inside the run and assign a value to it
run["seed"] = 0.42

# Log a series of values
from random import random

epochs = 10
offset = random() / 5

for epoch in range(epochs):
    acc = 1 - 2**-epoch - random() / (epoch + 1) - offset
    loss = 2**-epoch + random() / (epoch + 1) + offset

    run["accuracy"].append(acc)
    run["loss"].append(loss)

# Upload an image
run["single_image"].upload("Lenna_test_image.png")

# Download the MNIST dataset
import mnist

train_images = mnist.train_images()
train_labels = mnist.train_labels()

# Upload a series of images
from neptune.types import File

for i in range(10):
    run["image_series"].append(
        File.as_image(
            train_images[i] / 255
        ),  # You can upload arrays as images using the File.as_image() method
        name=f"{train_labels[i]}",
    )

# Stop the connection and synchronize the data with the Neptune servers
run.stop()
```