HEP ML Lab

An end-to-end framework for machine learning application in high energy physics

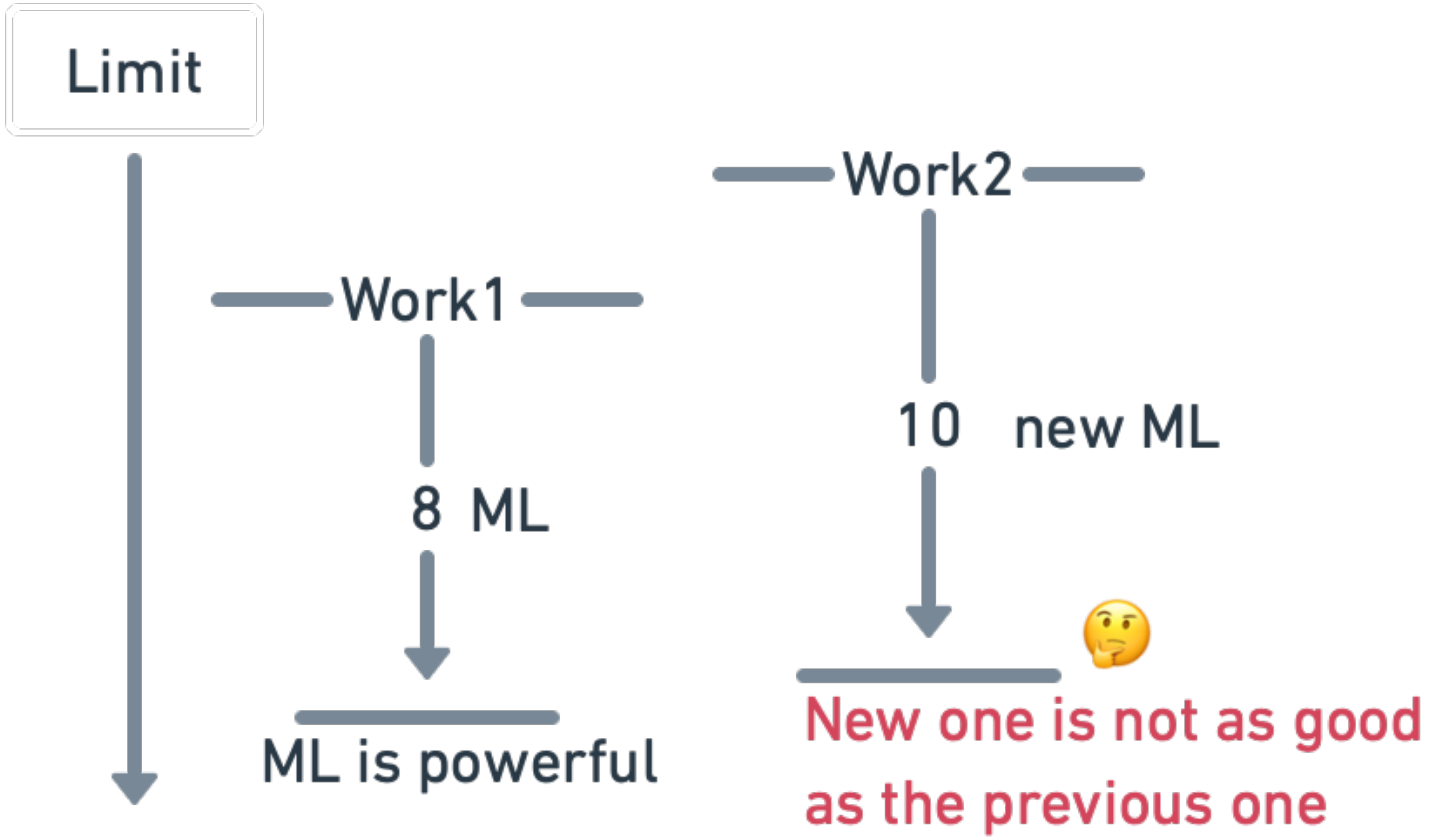
Jing Li, Hao Sun
Quantum Computing and Machine Learning Workshop (2023)

Table of contents

Introduction

Reproduction problems

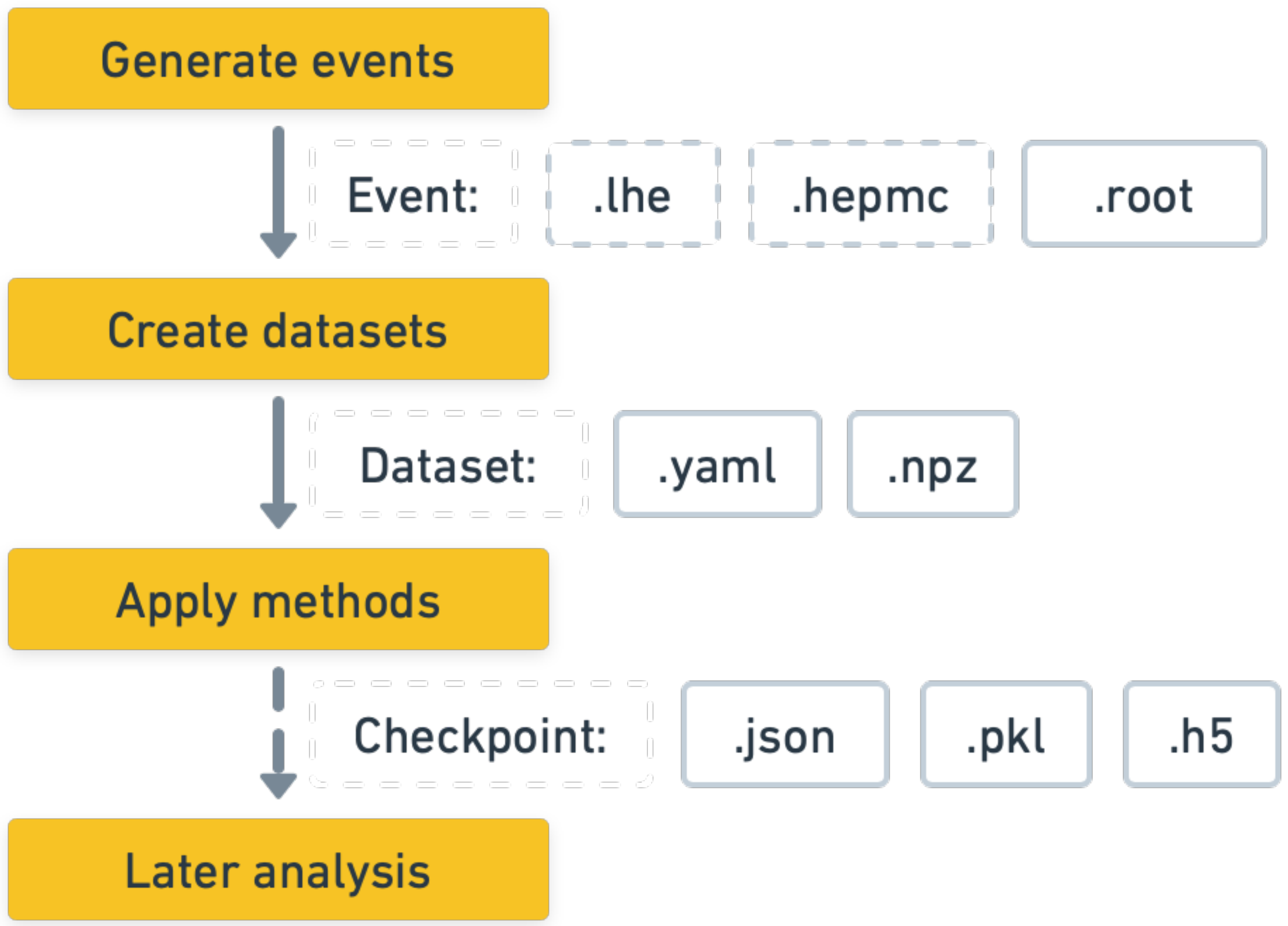
- A large amount of work explores the performance improvement brought by machine learning methods. Results are **promising** for the new physics search.
- However the lack of source codes makes it **quite difficult** to reproduce the results.
    - If they are generated under different conditions, at what extent we can say that new methods are truly powerful and worth to try in broader subjects?



Introduction

Control from end to end

- HEP ML Lab (HML) stands for high energy physics and machine learning.
- An end-to-end framework for applying machine learning into HEP studies.
    - Simplify the data flow: easier to manage and track different data.
    - Unify programming style across all stages: objected-oriented and Keras.
- All makes the results more reliable and reproducible.



Introduction

Comparison with others

Name	Data	Model	Style	Highlight
hep_ml	No	Yes	sklearn	Low correlation, theano-based, sklearn compatible
weaver	Yes	Yes	CLI + config	Support many dataset formats, config for all
JetNet	Yes	No	custom	Three datasets, generative evaluation metrics
pd4ml	Yes	Yes	modified Keras	Five datasets, model templates
MLAnalysis	Yes	Yes	custom	LHE/LHCO data, three ML algorithms
mapyde	Yes	No	CLI + TUI + config	Madgraph5 workflow, for a specific problem
madminer	Yes	Yes	custom	Madgraph5 workflow, for a specific problem
hep-ml-lab	Yes	Yes	Keras	Madgraph5 workflow, not specific for one problem

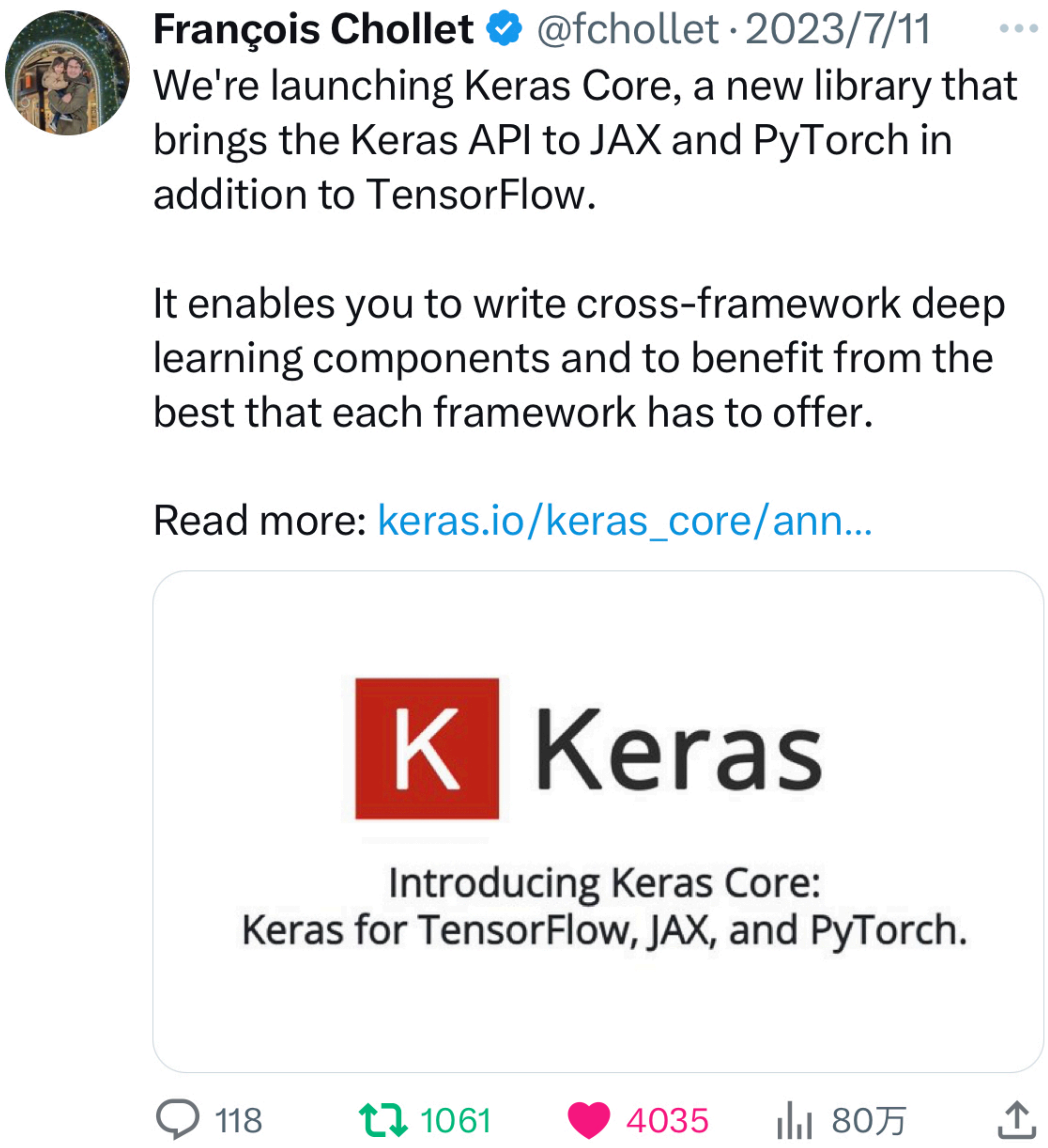
- Data: data generation and/or dataset interface support.
- Model: machine learning models and/or other methods.

Introduction

Comparison with others

- HEP ML Lab emphasizes:
  - data control from the very beginning, event generator.
  - concise and consistent training style of Keras

👉 Keras announces to be a multi-backend wrapper, bringing more possibility for the future.



Quick start

Generate events

- Let's first generate some events
    - Z boson to dijet as signal.
    - QCD jets as background.
- "generate events" and "add process" are "processes" now.
- "output", "launch" as usual.
- Check info after finishing a run.

```python
from hml.generators import Madgraph5

signal_generator = Madgraph5(
    executable="mg5_aMC",
    processes=["p p > z z, z > j j, z > ve ve~"],
    output="./data/pp2zz",
    shower="Pythia8",
    detector="Delphes",
    settings={
        "nevents": 10000,
        "iseed": 42,
        "htjmin": 400,
    },
)

signal_generator.launch()

sig_run = signal_generator.runs[0]
print(f"cross section (pb): {sig_run.cross_section}")
print(f"number of events: {sig_run.n_events}")
```

Quick start

Generate events

- Most parameters are moved into initialization so that **launch** starts generation immediately.

- Info is extracted from **print_results** command of MadEvents.

- Change settings and launch your next run. Get summary of all runs via **summary** method.

```
Generating events...
Running Pythia8...
Running Delphes...
Storing files...
Done
cross section (pb): 0.00077034
number of events: 10000
```

Processes: ['p p > z z, z > l+ l-, z > j j']

#	Name (N subruns)	Tag	Cross section +- Error pb	N events
0	run_01 (1)	tag_1	6.63090e-01 +- 8.13908e-03	100
1	run_02 (2)	tag_1	6.63090e-01 +- 8.13908e-03	100
2	run_03 (3)	tag_1	6.63334e-01 +- 3.99329e-03	300

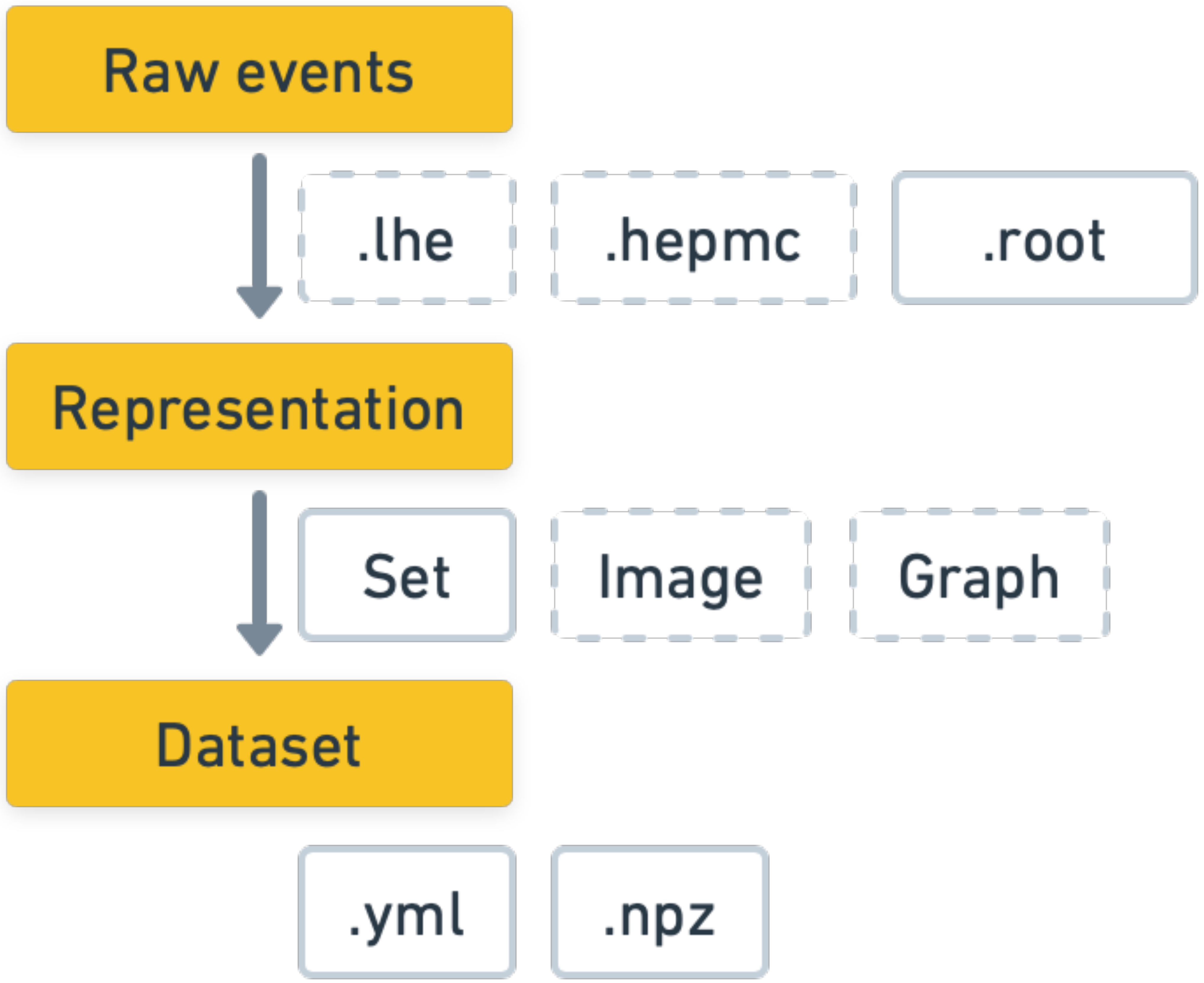
Output: data/pp2zz

print_results itself has bugs

Quick start

Create datasets

- Generator produces events in "raw" format.
- Data then is transformed into proper representation.
- Finally, combine data and labels into a dataset.

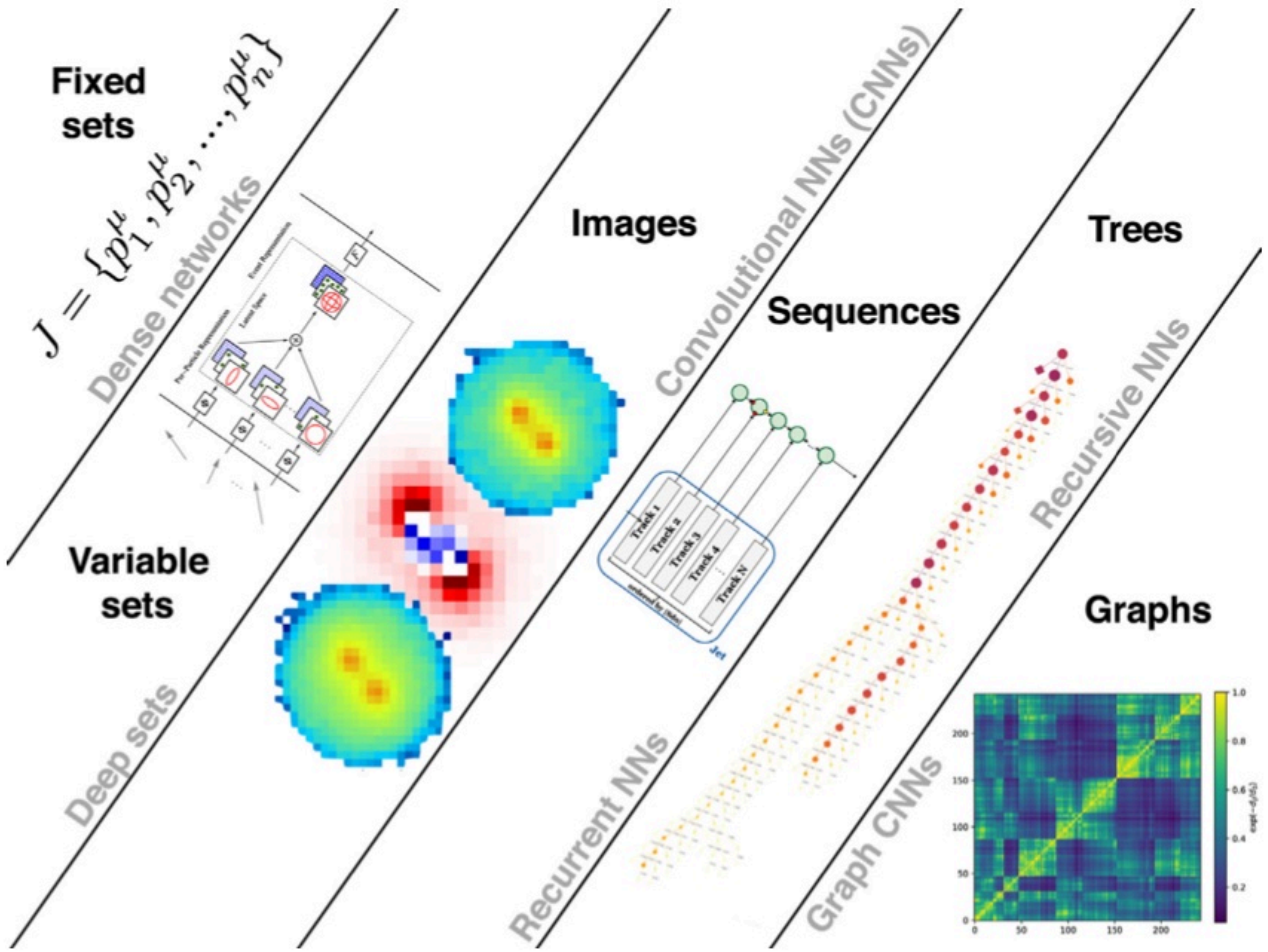


Quick start

Create datasets

- Set: 1D (N x F)
  - Use a set of observables to represent an event or a jet.
- Image: 3D (N x H x W x C)
  - Project particles onto a 2D plane.
- Graph: 2D (N x P x F)
  - Record particles' features.

[1709.04464] Jet Substructure at the Large Hadron Collider:
A Review of Recent Advances in Theory and Machine Learning



Quick start

Create datasets

- **MG5Run** links a launched run and resolve it to get info.
- We declare a set as defined: a set of **Observable**.
- Instead of indexing physics objects directly in event loop, short names like "Jet1" are used to find specific objects.

```python
from hml.generators import MG5Run
from hml.observables import DeltaR, M, Pt
from hml.representations import Set

sig_run = MG5Run("./data/pp2zz/Events/run_01/")
bkg_run = MG5Run("./data/pp2jj/Events/run_01/")

representation = Set(
    [
        Pt("Jet1"),
        Pt("Jet2"),
        DeltaR("Jet1", "Jet2"),
        M("FatJet1"),
    ]
)

```

Quick start

Create datasets

- Loop over events to fill the data and targets.
- This step aims to inject preselection to all events.
- To avoid time-consuming event loop, we're about to change backend from **PyROOT** to **Uproot** in later release.

```python
import numpy as np

data, target = [], []

for event in sig_run.events:
    if event.Jet_size >= 2 and event.FatJet_size >= 1:
        representation.from_event(event)
        data.append(representation.values)
        target.append(1)

for event in bkg_run.events:
    if event.Jet_size >= 2 and event.FatJet_size >= 1:
        representation.from_event(event)
        data.append(representation.values)
        target.append(0)

data = np.array(data, dtype=np.float32)
target = np.array(target, dtype=np.int32)

```

Quick start

Create datasets

- Complete the dataset with other information.
- **Dataset** is saved into two parts
    - metadata: Re-init a **Dataset** object.
    - dataset: Dataset value itself.

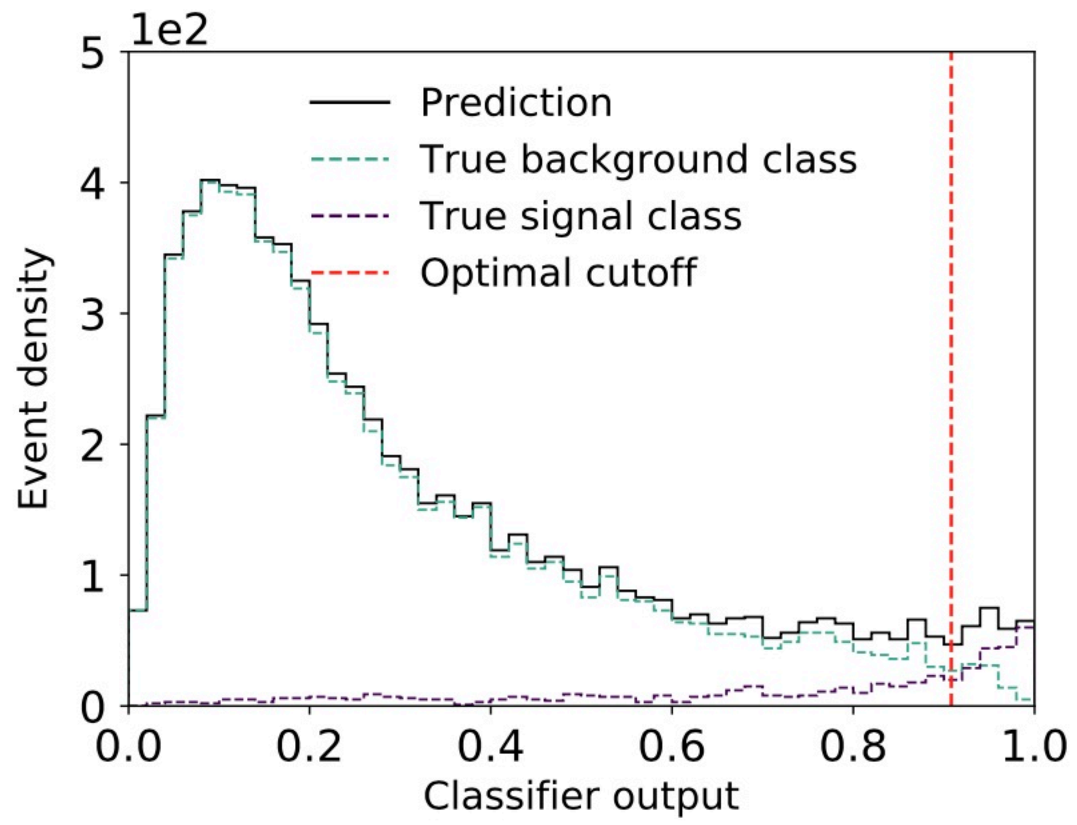
```python
from hml.datasets import Dataset

dataset = Dataset(
    data,
    target,
    feature_names=representation.names,
    target_names=["pp2jj", "pp2zz"],
    description="Demo dataset for Z vs QCD jets.",
    dir_path="./data/z_vs_qcd",
)
dataset.save(exist_ok=True)

```

Quick start

Apply methods

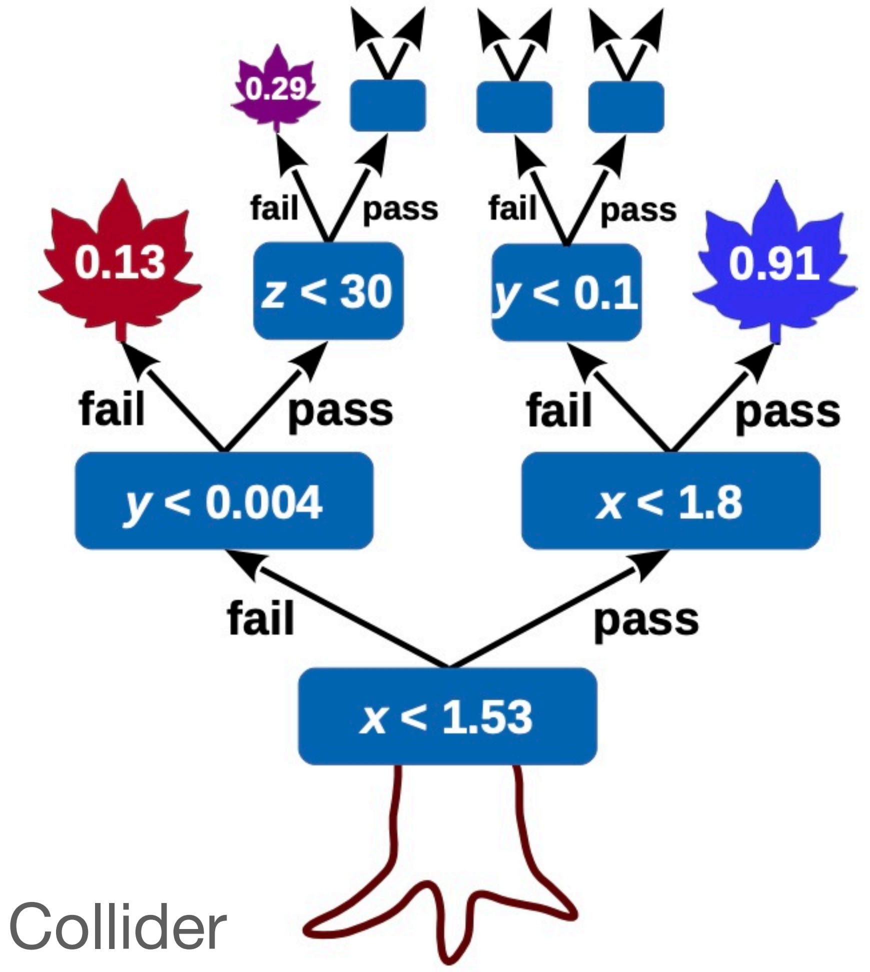
- Designed to contain three kinds of methods:
  - cut and count
  - tree
  - neural networks

[2108.03125] Beyond Cuts in Small Signal Scenarios

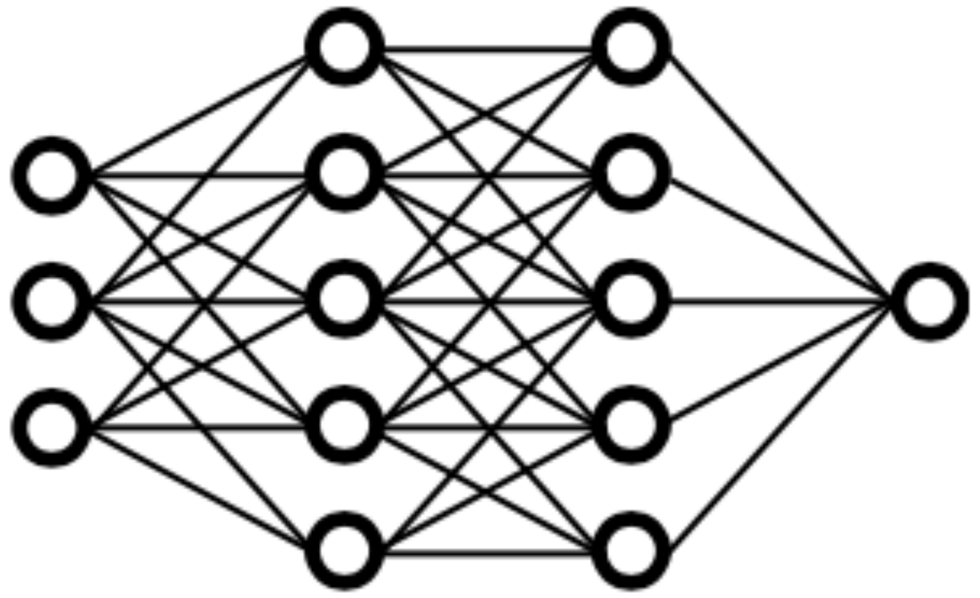


(a) XGBoost with optimized cutoff at 0.9081.

[2206.09645] Boosted decision trees



[1709.04464] Jet Substructure at the Large Hadron Collider



Quick start

Apply methods

init(...)	What does structure it have?
compile(optimizer, loss, metrics)	How to improve the weights?
fit(x, y, epochs...)	Other parameters of train process
evaluate(x, y)	How performant is it on other data?
summary()	Show model information
predict(x)	Given data, what is predicted value?

Method is the minimum wrapper of Keras **Model**.

Quick start

Apply methods

- **Load** dataset from previous saved location.
- Split train & test sets with fixed random seed.
- One-hot encoded label to enable following **Metrics**.

```python
from hml.datasets import Dataset
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
dataset = Dataset.load("./data/z_vs_qcd")

x_train, x_test, y_train, y_test = train_test_split(
    dataset.data,
    dataset.target,
    test_size=0.2,
    random_state=42,
)

# Convert the labels to categorical
y_train = to_categorical(y_train, dtype="int32")
y_test = to_categorical(y_test, dtype="int32")
```

Quick start

Apply methods

```python
from hml.methods import BoostedDecisionTree, CutAndCount, ToyMLP
from hml.metrics import MaxSignificance, RejectionAtEfficiency
from keras.losses import CategoricalCrossentropy
from keras.metrics import CategoricalAccuracy

```

- **MaxSignificance** calculates the maximum significance under uniform distributed thresholds.

$$\text{significance} = \sqrt{2\left((S+B)\ln\left(1+\frac{S}{B}\right)-S\right)}$$

- **RejectionAtEfficiency** ($1/\varepsilon_b$ at $\varepsilon_s = 50\,\%$) calculates the background rejection at a given signal efficiency.

Quick start

Apply methods

- Follow the training workflow
  - **Initialize** methods to define their structure respectively.
  - **Compile** each to determine how to improve itself and monitor performance.
  - **Fit** methods' weights on dataset.

```
m1 = BoostedDecisionTree(n_estimators=10)
m2 = CutAndCount(n_bins=100)
m3 = ToyMLP(input_shape=(x_train.shape[1],))

m1.compile(
    loss=CategoricalCrossentropy(),
    metrics=[
        CategoricalAccuracy(name="acc"),
        MaxSignificance(name="max_sig"),
        RejectionAtEfficiency(name="r50"),
    ],
)
m2.compile(...)
m3.compile(...)

m1.fit(x_train, y_train)
m2.fit(...)
m3.fit(...)

```

Quick start

Apply methods

```
Cut 1/4 - loss: 1.9366 - acc: 0.8798 - max_sig: 113.1778 - r50: 8.2616
Cut 2/4 - loss: 2.1924 - acc: 0.8719 - max_sig: 173.7675 - r50: 15.8622
Cut 3/4 - loss: 3.8445 - acc: 0.8351 - max_sig: 209.4424 - r50: 23.7669
Cut 4/4 - loss: 4.3686 - acc: 0.8086 - max_sig: 237.2822 - r50: 31.6540

Iter 1/10 - loss: 1.2097 - acc: 0.8795 - max_sig: 209.1248 - r50: 793.0361
Iter 2/10 - loss: 1.0733 - acc: 0.9162 - max_sig: 270.3814 - r50: 185.3986
Iter 3/10 - loss: 0.9599 - acc: 0.9328 - max_sig: 327.6703 - r50: 669.1434
...

Epoch 1/10
51/51 - 6s - loss: 0.9719 - acc: 0.8862 - max_sig: 186.6020 - r50: 31.5840 - 6s/epoch - 117ms/step
Epoch 2/10
51/51 - 1s - loss: 0.8845 - acc: 0.8881 - max_sig: 204.8537 - r50: 38.1710 - 1s/epoch - 23ms/step
Epoch 3/10
51/51 - 1s - loss: 0.7423 - acc: 0.8981 - max_sig: 209.3404 - r50: 44.6123 - 1s/epoch - 22ms/step
...
```

- Similar training histories. They can be retrieved by returned value of **fit**.

Quick start

Apply methods

```python
from tabulate import tabulate

results1 = method1.evaluate(x_test, y_test)
results2 = method2.evaluate(x_test, y_test)
results3 = method3.evaluate(x_test, y_test, verbose=2)
results = {}

results["name"] = [method1.name, method2.name, method3.name]
for k in results1.keys():
    results[k] = results1[k] + results2[k] + results3[k]

print("> Results:")
print(tabulate(results, headers="keys", floatfmt=".4f"))
```

```
> Results:
name                      loss     acc    max_sig       r50
----------------------  ------  ------  ---------  --------
boosted_decision_tree   0.2611  0.9586   601.7032  647.3771
cut_and_count           4.4163  0.8037   243.9667   33.6241
toy_mlp                 0.5475  0.9350   111.5401  444.2333
```

- **Evaluate** methods using metrics defined in **compile** methods. Could also compile once again to use other metrics.
- Later more metrics will be added to complete benchmark.

Future

Roadmap

- 0.2.x
  - Add random seed, batch run, auto tag
  - Change backend from PyROOT to Uproot
- 0.3.x
  - Support loading data from Zenodo, Hugging Face, GitHub, kaggle
  - Support image and graph representation and ToyCNN, ToyGNN to test
- 0.4.x protocol to keras

Future

Contribution

- HEP ML Lab itself contributes to Scikit-HEP.
    - Based on core packages of this community.
    - Support the principle of minimum dependency.
- We also welcome contributions from community.
    - Make your work reproducible more and more.
    - Currently, we are refactoring our work: [2303.15920] Probing Heavy Neutrinos at the LHC from Fat-jet using Machine Learning.

Thank YOU!