Training Samples Preparation

mo.jia@stonybrook.edu

Outline

- Framework
- Implementation on SeaWulf
- Test running results
- Next Steps

Framework

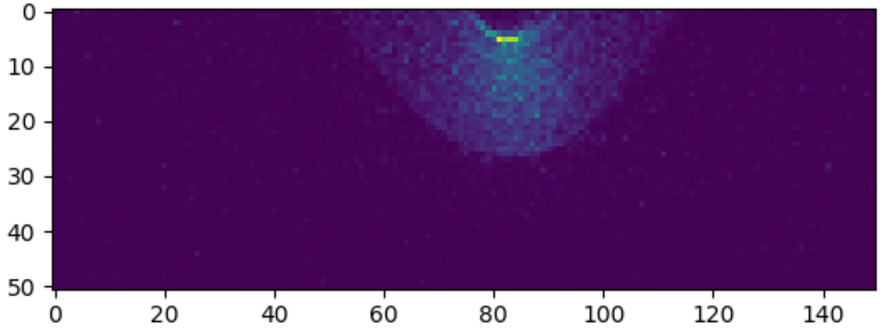
- Pyioopt
  - a python package to read WCSim root files and generate images of hits on the PMTs
  - Ultimately will be able to convert the data stored in root files to hdf5 format files, i.e. the training samples
  - Use pybind11 as the bindings to call c++ from python

Implementation

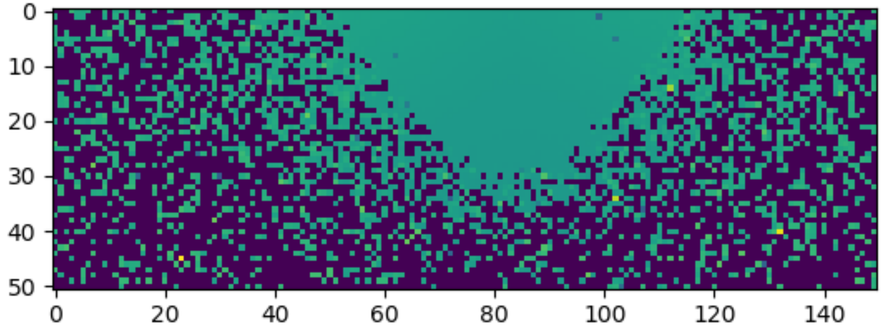
- Python3
- Root6
- WCSim
  - Set an environment variable ${WCSIMDIR}
- Append the directory of pyioopt to the list of ${PYTHONPATH}

Test Running Results

Muon events in one root file

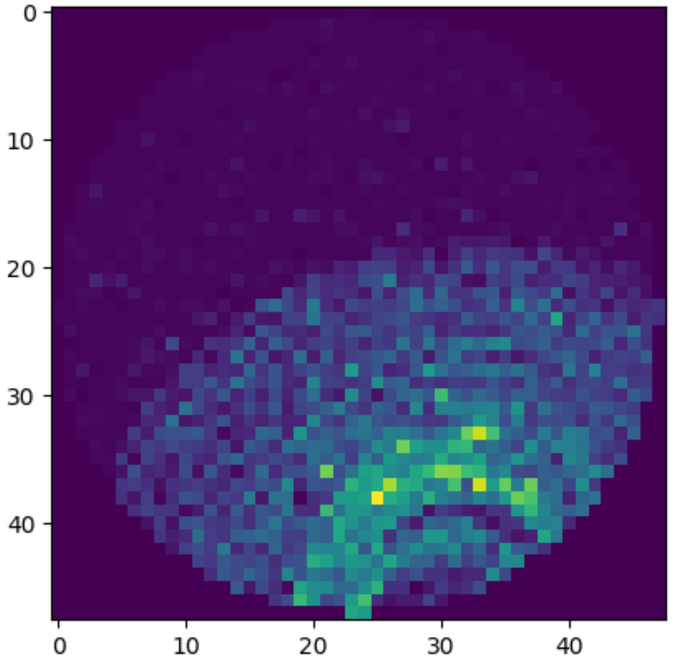


Barrel q

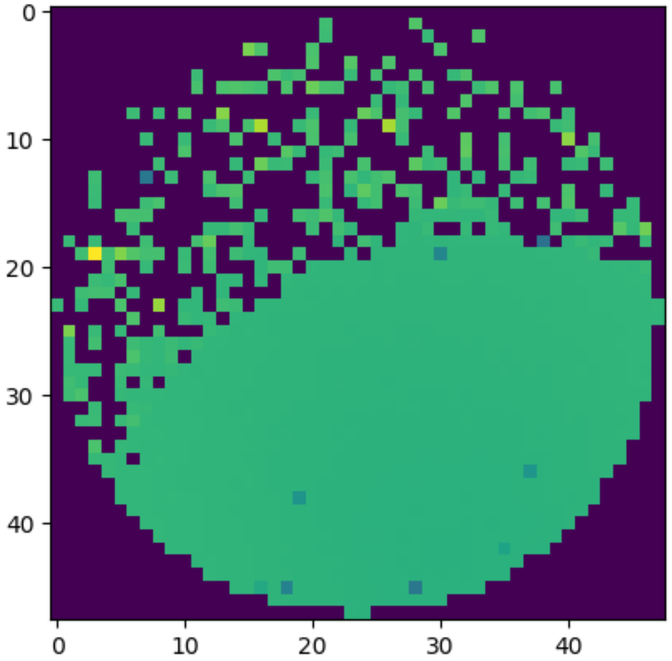


Barrel t

Test Running Results

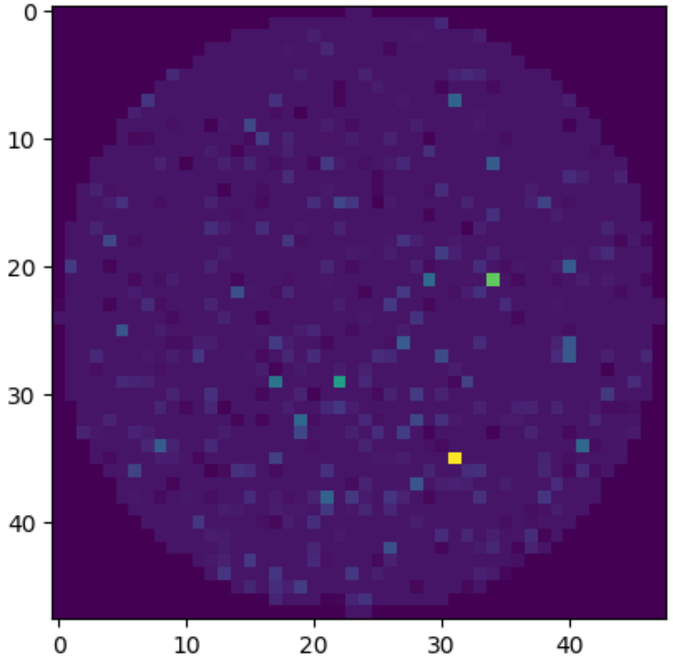


Bottom q

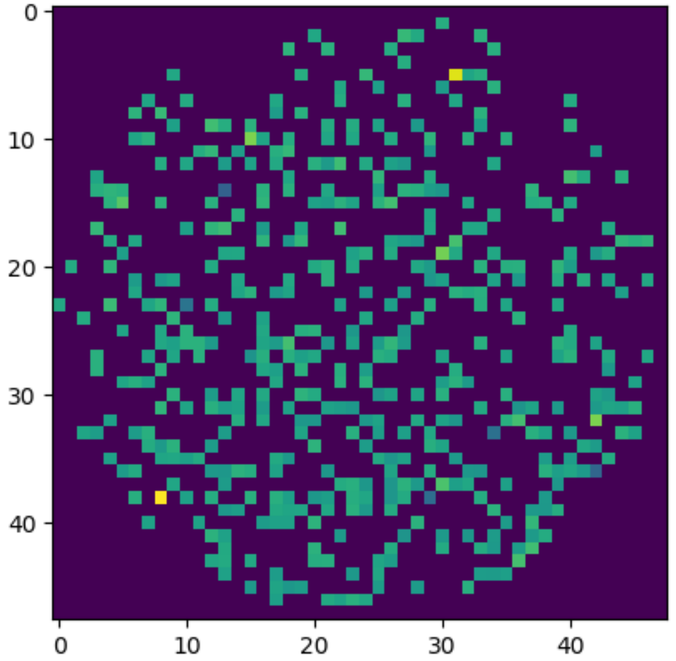


Bottom t

Test Running Results



Top q



Top t

Next Steps

- The ultimate goal is to convert the previous “images”, i.e. data on the grid, into hdf5 files with WatChMal format
- Study the DataTools package in WatChMal
- Learn more about hdf5 and the interface to it in python

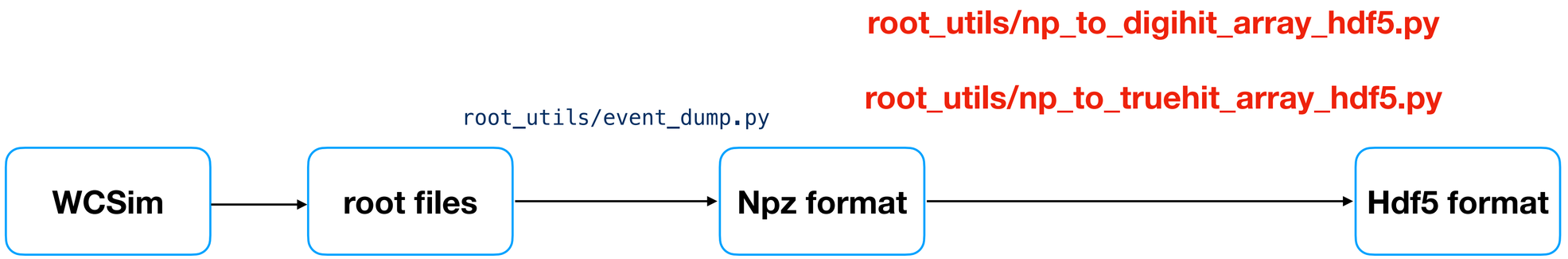
DataTools Package in WatChMal

DataTools Package

- Tools for production and manipulation of data for WatChMal
- Sub-directories:
  - data_quality
  - Visualization
  - cedar_scripts
  - root_utils

Data Production for WatChMal

Based on cedar_scripts/



run_WCSim_jobs.sh

make_digihit_h5.sh

make_truehit_h5.sh

NPZ is a file **format** by numpy that provides storage of array data using gzip compression.

root_utils/np_to_truehit_array_hdf5.py

- Create a h5py handle

```
20      config = get_args()
21      print("ouput file:", config.output_file)
22      f = h5py.File(config.output_file, 'w')
23
24      script_path = os.path.dirname(os.path.abspath(__file__))
25      git_status = subprocess.check_output(['git', '-C', script_path, 'status', '--porcelain', '--untracked-files=no']).decode()
26      if git_status:
27          raise Exception("Directory of this script ({}) is not a clean git directory:\n{}Need a clean git directory for storing script version in output file.".f
28      git_describe = subprocess.check_output(['git', '-C', script_path, 'describe', '--always', '--long', '--tags']).decode().strip()
29      print("git describe for path to this script ({}):".format(script_path), git_describe)
30      f.attrs['git-describe'] = git_describe
31      f.attrs['command'] = str(sys.argv)
32      f.attrs['timestamp'] = str(datetime.now())
33
```

root_utils/np_to_truehit_array_hdf5.py

- Get total event numbers and hit numbers

```
34      total_rows = 0
35      total_hits = 0
36      print("counting events and hits in files")
37      for input_file in config.input_files:
38          print(input_file, flush=True)
39          if not os.path.isfile(input_file):
40              raise ValueError(input_file+" does not exist")
41          npz_file = np.load(input_file)
42          hit_pmts = npz_file['true_hit_pmt']
43          total_rows += hit_pmts.shape[0]
44          for h in hit_pmts:
45              total_hits += h.shape[0]
46
47      print(len(config.input_files), "files with", total_rows, "events with ", total_hits, "hits")
```

total_rows = number of events
total_hits = number of hits

root_utils/np_to_truehit_array_hdf5.py

- Create datasets stored in the output file
  - Labels
  - root_files
  - event_ids
  - hit_time
  - hit_pmt
  - hit_parent
  - event_his_index
  - Energies
  - Positions
  - Angles
  - Veto
  - Veto2

```
49      dset_labels=f.create_dataset("labels",
50                                   shape=(total_rows,),
51                                   dtype=np.int32)
52      dset_PATHS=f.create_dataset("root_files",
53                                  shape=(total_rows,),
54                                  dtype=h5py.special_dtype(vlen=str))
55      dset_IDX=f.create_dataset("event_ids",
56                                shape=(total_rows,),
57                                dtype=np.int32)
58      dset_hit_time=f.create_dataset("hit_time",
59                                     shape=(total_hits, ),
60                                     dtype=np.float32)
61      dset_hit_pmt=f.create_dataset("hit_pmt",
62                                    shape=(total_hits, ),
63                                    dtype=np.int32)
64      dset_hit_parent=f.create_dataset("hit_parent",
65                                       shape=(total_hits, ),
66                                       dtype=np.int32)
67      dset_event_hit_index=f.create_dataset("event_hits_index",
68                                            shape=(total_rows,),
69                                            dtype=np.int64) # int32 is too small to fit large indices
70      dset_energies=f.create_dataset("energies",
71                                     shape=(total_rows, 1),
72                                     dtype=np.float32)
73      dset_positions=f.create_dataset("positions",
74                                      shape=(total_rows, 1, 3),
75                                      dtype=np.float32)
76      dset_angles=f.create_dataset("angles",
77                                   shape=(total_rows, 2),
78                                   dtype=np.float32)
79      dset_veto = f.create_dataset("veto",
80                                   shape=(total_rows,),
81                                   dtype=np.bool_)
82      dset_veto2 = f.create_dataset("veto2",
83                                    shape=(total_rows,),
84                                    dtype=np.bool_)
```

root_utils/np_to_truehit_array_hdf5.py

- Read in data from npz file by file and set the values of datasets

```
    offset = 0
    offset_next = 0
    hit_offset = 0
    hit_offset_next = 0
    label_map = {22: 0, 11: 1, 13: 2}
    for input_file in config.input_files:
        print(input_file, flush=True)
        npz_file = np.load(input_file, allow_pickle=True)
        good_events = ~np.isnan(file_event_triggers[input_file])
        event_triggers = file_event_triggers[input_file][good_events]
        event_ids = npz_file['event_id'][good_events]
        root_files = npz_file['root_file'][good_events]
        pids = npz_file['pid'][good_events]
        positions = npz_file['position'][good_events]
        directions = npz_file['direction'][good_events]
        energies = npz_file['energy'][good_events]
        hit_times = npz_file['digi_hit_time'][good_events]
        hit_charges = npz_file['digi_hit_charge'][good_events]
        hit_pmts = npz_file['digi_hit_pmt'][good_events]
        hit_triggers = npz_file['digi_hit_trigger'][good_events]
        track_pid = npz_file['track_pid'][good_events]
        track_energy = npz_file['track_energy'][good_events]
        track_stop_position = npz_file['track_stop_position'][good_events]
        track_start_position = npz_file['track_start_position'][good_events]


        offset_next += event_ids.shape[0]

        dset_IDX[offset:offset_next] = event_ids
        dset_PATHS[offset:offset_next] = root_files
        dset_energies[offset:offset_next,:] = energies.reshape(-1,1)
        dset_positions[offset:offset_next,:,:] = positions.reshape(-1,1,3)

        labels = np.full(pids.shape[0], -1)
        for l, v in label_map.items():
            labels[pids==l] = v
        dset_labels[offset:offset_next] = labels

        polars = np.arccos(directions[:,1])
        azimuths = np.arctan2(directions[:,2], directions[:,0])
        dset_angles[offset:offset_next,:] = np.hstack((polars.reshape(-1,1),azimuths.reshape(-1,1)))
```

continued

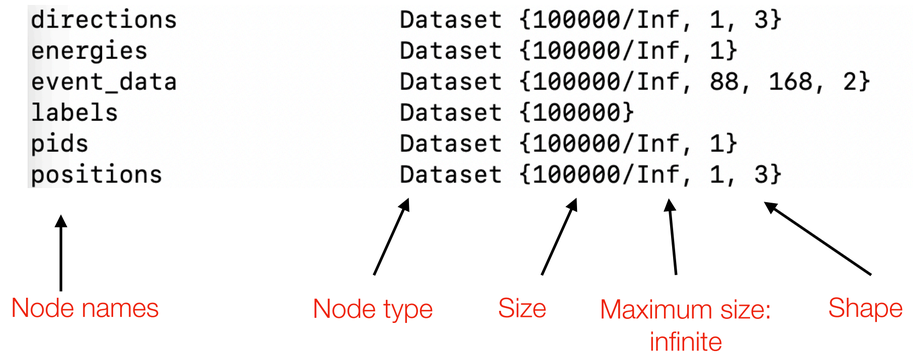
```
        for i, (pids, energies, starts, stops) in enumerate(zip(track_pid, track_energy,track_start_position, track_stop_position)):
            muons_above_threshold = (np.abs(pids) == 13) & (energies > 166)
            electrons_above_threshold = (np.abs(pids) == 11) & (energies > 2)
            gammas_above_threshold = (np.abs(pids) == 22) & (energies > 2)
            above_threshold = muons_above_threshold | electrons_above_threshold | gammas_above_threshold
            outside_tank = (np.linalg.norm(stops[:,(0,2)], axis=1) > config.radius) | (np.abs(stops[:, 1]) > config.half_height)
            dset_veto[offset+i] = np.any(above_threshold & outside_tank)
            end_energy_estimate = energies - np.linalg.norm(stops - starts, axis=1)*2
            muons_above_threshold = (np.abs(pids) == 13) & (end_energy_estimate > 166)
            electrons_above_threshold = (np.abs(pids) == 11) & (end_energy_estimate > 2)
            gammas_above_threshold = (np.abs(pids) == 22) & (end_energy_estimate > 2)
            above_threshold = muons_above_threshold | electrons_above_threshold | gammas_above_threshold
            dset_veto2[offset+i] = np.any(above_threshold & outside_tank)

        for i, (trigs, times, charges, pmts) in enumerate(zip(hit_triggers, hit_times, hit_charges, hit_pmts)):
            dset_event_hit_index[offset+i] = hit_offset
            hit_indices = np.where(trigs==event_triggers[i])[0]
            hit_offset_next += len(hit_indices)
            dset_hit_time[hit_offset:hit_offset_next] = times[hit_indices]
            dset_hit_charge[hit_offset:hit_offset_next] = charges[hit_indices]
            dset_hit_pmt[hit_offset:hit_offset_next] = pmts[hit_indices]
            hit_offset = hit_offset_next

        offset = offset_next
```

Comparison with Current h5 files on Ivy

- Example: IWCDgrid_varyAll_mu-_20-2000MeV_100k.h5
- datasets stored in the file:
  - Directions shape (1,3)
  - Energies shape (1,)
  - event_data shape (88,168,2)
  - Labels shape ?
  - Pid shape (1,)
  - Positions shape (1, 3)



Next Steps

- Look into the codes in root_utils/event_dump.py
- Learn more about hdf5 and the interface h5py
- NumPy