Neuromorphic Datasets Processing

Authors: fangwei123456

`spikingjelly.datasets` provides frequently-used neuromorphic datasets, including N-MNIST [1], CIFAR10-DVS [2], DVS128 Gesture [3], N-Caltech101 [1], ASLDVS [4], etc. All datasets are processed by SpikingJelly in the same method, which is friendly for developers to write codes for new datasets. In this tutorial, we will take DVS 128 Gesture dataset as an example to show how to use SpikingJelly to process neuromorphic datasets.

Download Automatically/Manually

SpikingJelly can download some datasets (e.g., CIFAR10-DVS) automatically. When we first use these datasets, SpikingJelly will download the dataset to `download` in the root directory. The `downloadable()` function of each dataset defines whether this dataset can be downloaded automatically, and the `resource_url_md5()` function defines the download url and MD5 of each file. Here is an example:

```
from spikingjelly.datasets.cifar10_dvs import CIFAR10DVS
from spikingjelly.datasets.dvs128_gesture import DVS128Gesture

print('CIFAR10-DVS downloadable', CIFAR10DVS.downloadable())
print('resource, url, md5/n', CIFAR10DVS.resource_url_md5())

print('DVS128Gesture downloadable', DVS128Gesture.downloadable())
print('resource, url, md5/n', DVS128Gesture.resource_url_md5())
```

The outputs are:

```
CIFAR10-DVS downloadable True
resource, url, md5
 [('airplane.zip', 'https://ndownloader.figshare.com/files/7712788',
'0afd5c4bf9ae06af762a77b180354fdd'), ('automobile.zip',
'https://ndownloader.figshare.com/files/7712791', '8438dfeba3bc970c94962d995b1b9bdd'),
('bird.zip', 'https://ndownloader.figshare.com/files/7712794',
'a9c207c91c55b9dc2002dc21c684d785'), ('cat.zip',
'https://ndownloader.figshare.com/files/7712812', '52c63c677c2b15fa5146a8daf4d56687'),
('deer.zip', 'https://ndownloader.figshare.com/files/7712815',
'b6bf21f6c04d21ba4e23fc3e36c8a4a3'), ('dog.zip',
'https://ndownloader.figshare.com/files/7712818', 'f379ebdf6703d16e0a690782e62639c3'),
('frog.zip', 'https://ndownloader.figshare.com/files/7712842',
'cad6ed91214b1c7388a5f6ee56d08803'), ('horse.zip',
'https://ndownloader.figshare.com/files/7712851', 'e7cbbf77bec584ffbf913f00e682782a'),
('ship.zip', 'https://ndownloader.figshare.com/files/7712836',
'41c7bd7d6b251be82557c6cce9a7d5c9'), ('truck.zip',
'https://ndownloader.figshare.com/files/7712839', '89f3922fd147d9aeff89e76a2b0b70a7')]
DVS128Gesture downloadable False
resource, url, md5
 [('DvsGesture.tar.gz',
'https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794',
'8a5c71fb11e24e5ca5b11866ca6c00a1'), ('gesture_mapping.csv',
'https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794',
'109b2ae64a0e1f3ef535b18ad7367fd1'), ('LICENSE.txt',
'https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794',
'065e10099753156f18f51941e6e44b66'), ('README.txt',
'https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794',
'a0663d3b1d8307c329a43d949ee32d19')]
```

The DVS128 Gesture dataset can not be downloaded automatically. But its `resource_url_md5()` will tell the user where to download. The DVS128 Gesture dataset can be downloaded from https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794. The box website does not allow us to download data by python codes without login. Thus, the user has to download it manually. Suppose we have downloaded the dataset into `E:/datasets/DVS128Gesture/download`, then the directory structure is

```
.
|-- DvsGesture.tar.gz
|-- LICENSE.txt
|-- README.txt
`-- gesture_mapping.csv
```

Note

Different frameworks may use different pre-processing methods on the DVS128 Gesture dataset and cause different samples number. Refer to the API doc of `spikingjelly.datasets.dvs128_gesture.DVS128Gesture` for more details.

latest

Get Events Data

Let us create a train set. We set `data_type='event'` to use Event data rather than frame data.

```
from spikingjelly.datasets.dvs128_gesture import DVS128Gesture

root_dir = 'D:/datasets/DVS128Gesture'
train_set = DVS128Gesture(root_dir, train=True, data_type='event')
```

SpikingJelly will do the followed work when running these codes:

1. Check whether the dataset exists. If the dataset exists, check MD5 to ensure the dataset is complete. Then SpikingJelly will extract the original data into the `extracted` folder
2. The sample in DVS128 Gesture is the video that records one actor displaying different gestures under different illumination conditions. Hence, an AER sample contains many gestures and there is also an adjoint csv file to label the time stamp of each gesture. Hence, an AER sample is not a sample with one class but multi-classes. SpikingJelly will use multi-threads to cut and extract each gesture from these files.

Here are the terminal outputs:

```
The [D:/datasets/DVS128Gesture/download] directory for saving downloaded files already exists,
check files...
Mkdir [D:/datasets/DVS128Gesture/extract].
Extract [D:/datasets/DVS128Gesture/download/DvsGesture.tar.gz] to
[D:/datasets/DVS128Gesture/extract].
Mkdir [D:/datasets/DVS128Gesture/events_np].
Start to convert the origin data from [D:/datasets/DVS128Gesture/extract] to
[D:/datasets/DVS128Gesture/events_np] in np.ndarray format.
Mkdir [('D:/datasets/DVS128Gesture//events_np//train',
'D:/datasets/DVS128Gesture//events_np//test').
Mkdir ['0', '1', '10', '2', '3', '4', '5', '6', '7', '8', '9'] in
[D:/datasets/DVS128Gesture/events_np/train] and ['0', '1', '10', '2', '3', '4', '5', '6', '7',
'8', '9'] in [D:/datasets/DVS128Gesture/events_np/test].
Start the ThreadPoolExecutor with max workers = [8].
Start to split [D:/datasets/DVS128Gesture/extract/DvsGesture/user02_fluorescent.aedat] to
samples.
[D:/datasets/DVS128Gesture/events_np/train/0/user02_fluorescent_0.npz] saved.
[D:/datasets/DVS128Gesture/events_np/train/1/user02_fluorescent_0.npz] saved.

......

[D:/datasets/DVS128Gesture/events_np/test/8/user29_lab_0.npz] saved.
[D:/datasets/DVS128Gesture/events_np/test/9/user29_lab_0.npz] saved.
[D:/datasets/DVS128Gesture/events_np/test/10/user29_lab_0.npz] saved.
Used time = [1017.27s].
All aedat files have been split to samples and saved into
[('D:/datasets/DVS128Gesture//events_np//train',
'D:/datasets/DVS128Gesture//events_np//test')].
```

latest

We have to wait for a moment because the cutting and extracting are very slow. A `events_np` folder will be created and contain the train/test set:

```
|-- events_np
|   |-- test
|   `-- train
```

Print a sample:

```
event, label = train_set[0]
for k in event.keys():
    print(k, event[k])
print('label', label)
```

The output is:

```
t [80048267 80048277 80048278 ... 85092406 85092538 85092700]
x [49 55 55 ... 60 85 45]
y [82 92 92 ... 96 86 90]
p [1 0 0 ... 1 0 0]
label 0
```

where `event` is a dictionary with keys `['t', 'x', 'y', 'p']`;``label`` is the label of the sample. Note that the class number of DVS128 Gesture is 11.

Get Frames Data

The event-to-frame integrating method for pre-processing neuromorphic datasets is widely used. We use the same method from [5] in SpikingJelly. Data in neuromorphic datasets are in the formulation of $E(x_i, y_i, t_i, p_i)$ that represent the event's coordinate, time and polarity. We split the event's number N into T slices with nearly the same number of events in each slice and integrate events to frames. Note that T is also the simulating time-step. Denote a two channels frame as $F(j)$ and a pixel at (p, x, y) as $F(j, p, x, y)$, the pixel value is integrated from the events data whose indices are between j_l and j_r:

$$
\begin{aligned}
j_l &= \left\lfloor \frac{N}{T} \right\rfloor \cdot j \\
j_r &= \begin{cases} \left\lfloor \frac{N}{T} \right\rfloor \cdot (j+1), & \text{if } j < T-1 \\ N, & \text{if } j = T-1 \end{cases} \\
F(j, p, x, y) &= \sum_{i=j_l}^{j_r - 1} \mathcal{I}_{p,x,y}(p_i, x_i, y_i)
\end{aligned}
$$

latest

where $\lfloor \cdot \rfloor$ is the floor operation, $\mathcal{I}_{p,x,y}(p_i, x_i, y_i)$ is an indicator function and it equals 1 only when $(p, x, y) = (p_i, x_i, y_i)$.

SpikingJelly will integrate events to frames when running the followed codes:

```
train_set = DVS128Gesture(root_dir, train=True, data_type='frame', frames_number=20,
split_by='number')
```

The outputs from the terminal are:

```
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/0].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/1].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/10].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/2].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/3].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/4].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/5].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/6].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/7].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/8].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/9].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/0].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/1].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/10].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/2].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/3].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/4].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/5].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/6].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/7].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/8].
Mkdir [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/9].
Start ThreadPoolExecutor with max workers = [8].
Start to integrate [D:/datasets/DVS128Gesture/events_np/test/0/user24_fluorescent_0.npz] to
frames and save to [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/0].
Start to integrate [D:/datasets/DVS128Gesture/events_np/test/0/user24_fluorescent_led_0.npz]
to frames and save to [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/test/0].

......

Frames [D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/9/user23_lab_0.npz]
saved.Frames
[D:/datasets/DVS128Gesture/frames_number_20_split_by_number/train/9/user23_led_0.npz] saved.

Used time = [102.11s].
```

A `frames_number_20_split_by_number` folder will be created and contain the Frame data

latest

Print a sample:

```
frame, label = train_set[0]
print(frame.shape)
```

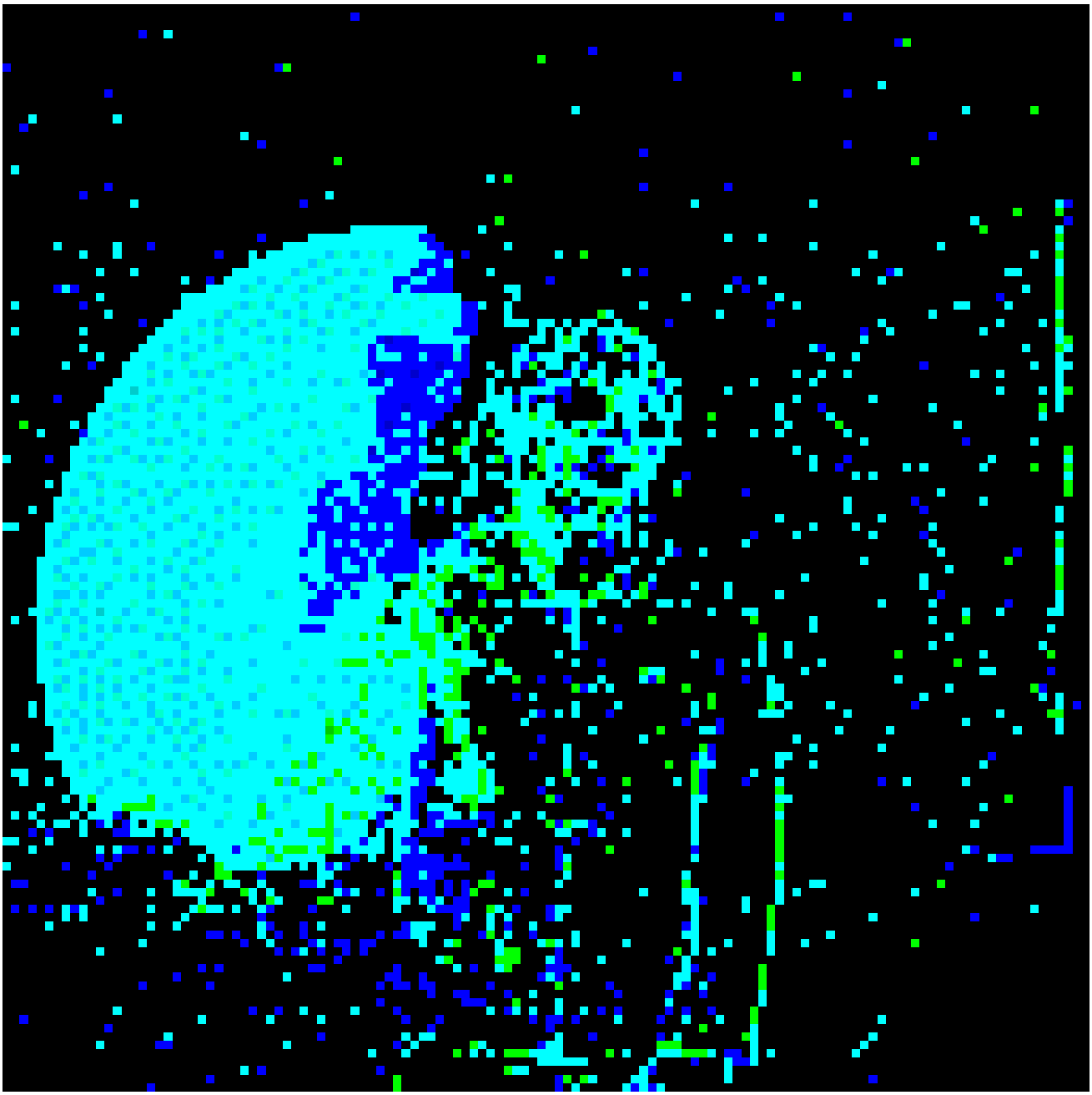
The outputs are:

```
(20, 2, 128, 128)
```

Let us visualize a sample:

```
from spikingjelly.datasets import play_frame
frame, label = train_set[500]
play_frame(frame)
```

We will get the images like:



Fixed Duration Integrating

Integrating by fixed duration is more compatible with the practical application. For example, if we set duration as `10 ms`, then a sample with length `L ms` can be integrated to frames with frame number `math.floor(L / 10)`. However, the lengths of samples in neuromorphic datasets are not identical, and we will get frames with different frame numbers when integrating with a fixed duration. Fortunately, we can use `spikingjelly.datasets.pad_sequence_collate` and `spikingjelly.datasets.padded_sequence_mask` to pad/unpad frames.

Example codes:

```
import torch
from torch.utils.data import DataLoader
from spikingjelly.datasets import pad_sequence_collate, padded_sequence_mask, dvs128_gesture
root='D:/datasets/DVS128Gesture'
train_set = dvs128_gesture.DVS128Gesture(root, data_type='frame', duration=1000000,
train=True)
for i in range(5):
    x, y = train_set[i]
    print(f'x[{i}].shape=[T, C, H, W]={x.shape}')
train_data_loader = DataLoader(train_set, collate_fn=pad_sequence_collate, batch_size=5)
for x, y, x_len in train_data_loader:
    print(f'x.shape=[N, T, C, H, W]={tuple(x.shape)}')
    print(f'x_len={x_len}')
    mask = padded_sequence_mask(x_len)  # mask.shape = [T, N]
    print(f'mask=\n{mask.t().int()}')
    break
```

The outputs are:

```
The directory [D:/datasets/DVS128Gesture\duration_1000000] already exists.
x[0].shape=[T, C, H, W]=(6, 2, 128, 128)
x[1].shape=[T, C, H, W]=(6, 2, 128, 128)
x[2].shape=[T, C, H, W]=(5, 2, 128, 128)
x[3].shape=[T, C, H, W]=(5, 2, 128, 128)
x[4].shape=[T, C, H, W]=(7, 2, 128, 128)
x.shape=[N, T, C, H, W]=(5, 7, 2, 128, 128)
x_len=tensor([6, 6, 5, 5, 7])
mask=
tensor([[1, 1, 1, 1, 1, 1, 0],
        [1, 1, 1, 1, 1, 1, 0],
        [1, 1, 1, 1, 1, 0, 0],
        [1, 1, 1, 1, 1, 0, 0],
        [1, 1, 1, 1, 1, 1, 1]], dtype=torch.int32)
```

Custom Integrating Method

SpikingJelly provides user-defined integrating method. The user should provide a function `custom_integrate_function` and the name of directory `custom_integrated_frames_dir_name` for saving frames.

`custom_integrate_function` is a user-defined function that inputs are `events, H, W`. `events` is a dict whose keys are `['t', 'x', 'y', 'p']` and values are `numpy.ndarray`. `H` is the height of the data and `W` is the weight of the data. For example, H=128 and W=128 for the DVS128 Gesture dataset. The function should return frames.

`custom_integrated_frames_dir_name` can be `None`, and then the name of directory for saving frames will be set to `custom_integrate_function.__name__`.

latest

For example, if we want to split events to two parts randomly, and integrate t
.
frames, we can define such a function:

```
import spikingjelly.datasets as sjds
def integrate_events_to_2_frames_randomly(events: Dict, H: int, W: int):
    index_split = np.random.randint(low=0, high=events['t'].__len__())
    frames = np.zeros([2, 2, H, W])
    t, x, y, p = (events[key] for key in ('t', 'x', 'y', 'p'))
    frames[0] = sjds.integrate_events_segment_to_frame(x, y, p, H, W, 0, index_split)
    frames[1] = sjds.integrate_events_segment_to_frame(x, y, p, H, W, index_split,
events['t'].__len__())
    return frames
```

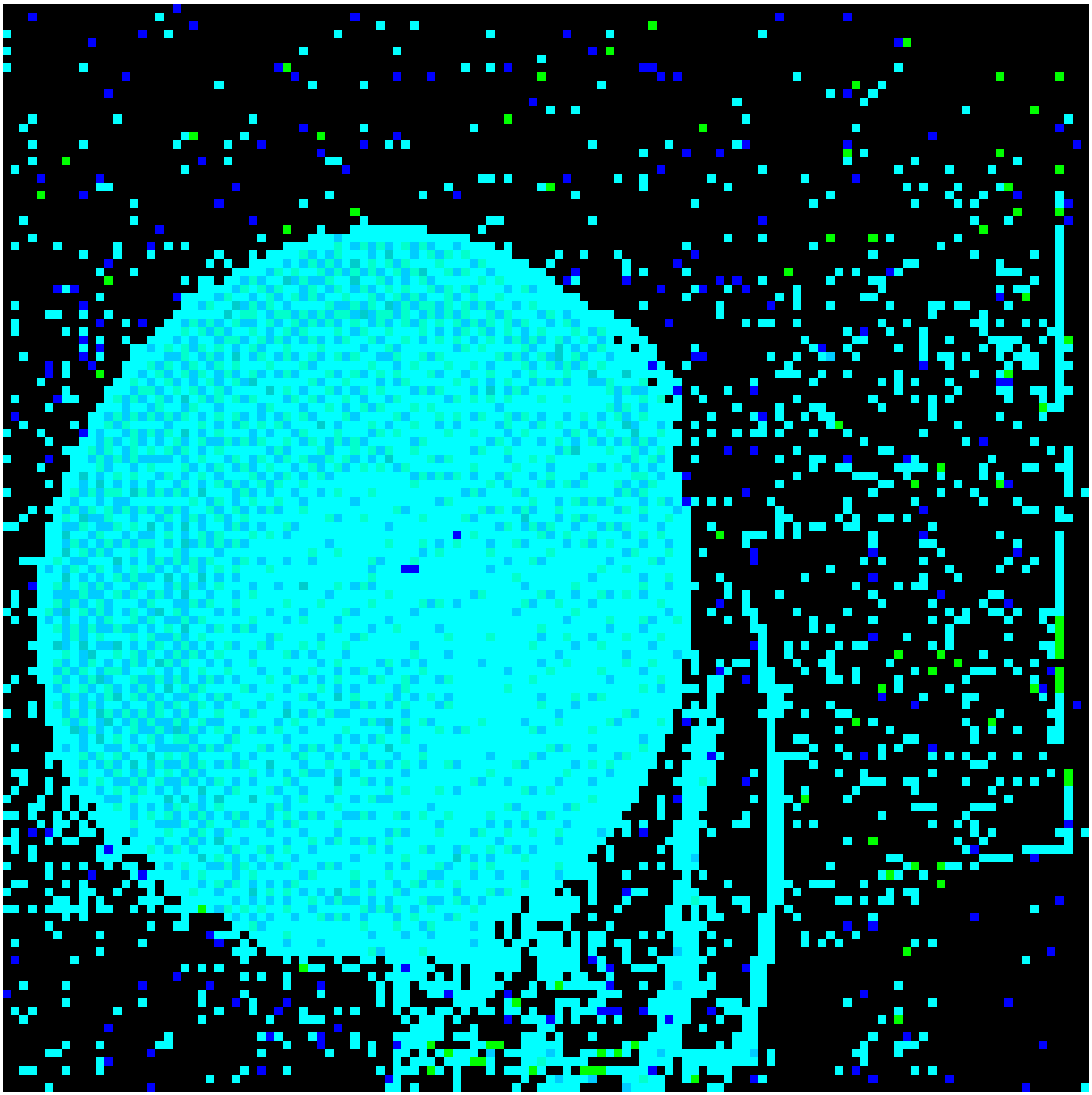
Now let us use this function to create a frames dataset:

```
train_set = DVS128Gesture(root_dir, train=True, data_type='frame',
custom_integrate_function=integrate_events_to_2_frames_randomly)
```

After the process is finished, there will be a `integrate_events_to_2_frames_randomly` directory in `root_dir`. And the `integrate_events_to_2_frames_randomly` directory will save our frames integrated by the custom integrating function.

Now let us visualize the frames:

```
from spikingjelly.datasets import play_frame
frame, label = train_set[500]
play_frame(frame)
```



SpikingJelly provides more methods to integrate events to frames. Read the API doc for more details.

[1] *(1, 2)* Orchard, Garrick, et al. “Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades.” Frontiers in Neuroscience, vol. 9, 2015, pp. 437–437.

[2] Li, Hongmin, et al. “CIFAR10-DVS: An Event-Stream Dataset for Object Classification.” Frontiers in Neuroscience, vol. 11, 2017, pp. 309–309.

[3] Amir, Arnon, et al. “A Low Power, Fully Event-Based Gesture Recognition System.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 7388–7397.

[4] Bi, Yin, et al. “Graph-Based Object Classification for Neuromorphic Vision Sensing.” 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 491–501.

[5] Fang, Wei, et al. “Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks.” ArXiv: Neural and Evolutionary Computing, 2020.

latest