Classify DVS Gesture

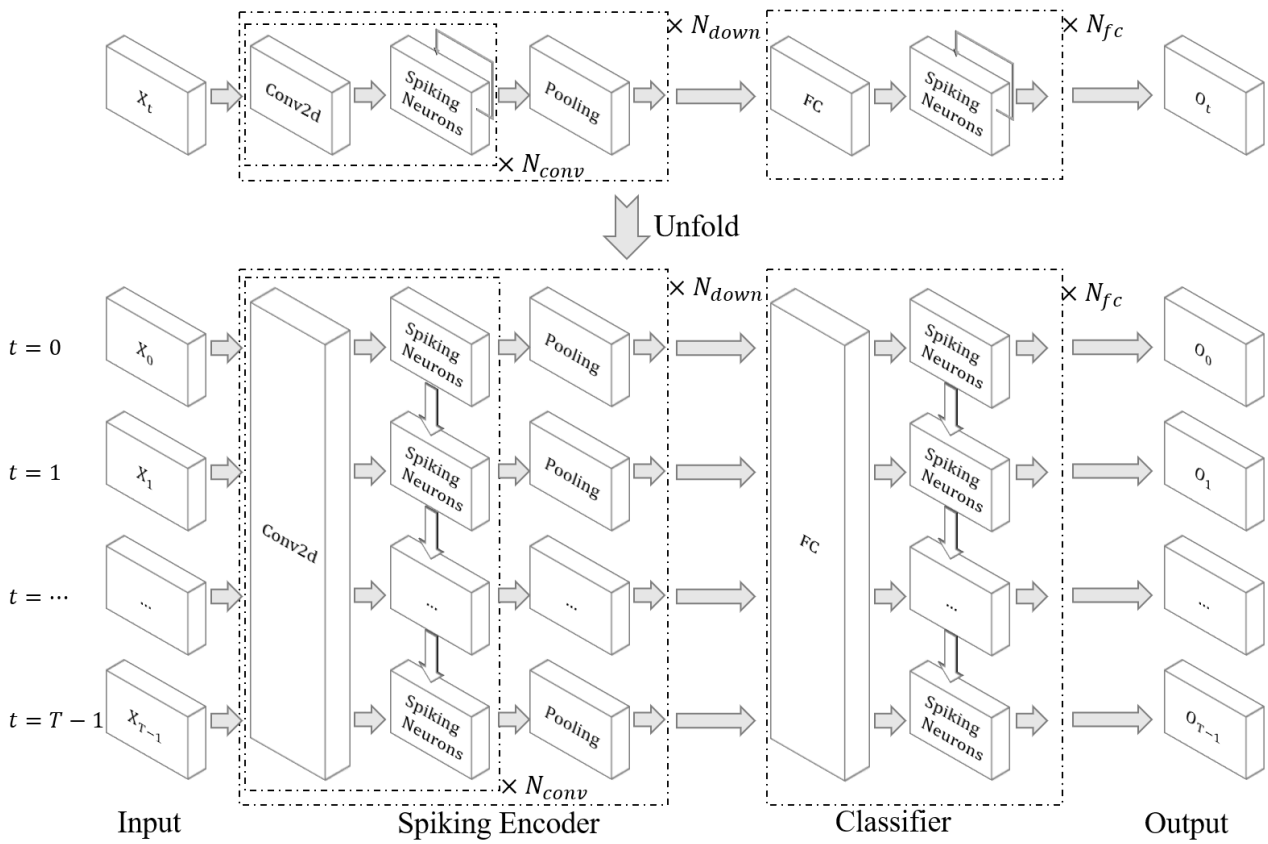
Author: fangwei123456

Translator: Qiu Haonan, fangwei123456

In Neuromorphic Datasets Processing, we have learned how to use neuromorphic datasets. Let's build a SNN to classify them.

Network Structure

We will use the network defined in [1], which has the following structure:



All networks in [1] are defined in `spikingjelly.activation_based.model.parametric_lif_net`. The network structure for DVS Gesture is:

```
# spikingjelly.activation_based.model.parametric_lif_net

import torch
import torch.nn as nn
from .. import layer

class DVSGestureNet(nn.Module):
    def __init__(self, channels=128, spiking_neuron: callable = None, *args, **kwargs):
        super().__init__()

        conv = []
        for i in range(5):
            if conv.__len__() == 0:
                in_channels = 2
            else:
                in_channels = channels

            conv.append(layer.Conv2d(in_channels, channels, kernel_size=3, padding=1,
bias=False))
            conv.append(layer.BatchNorm2d(channels))
            conv.append(spiking_neuron(*args, **kwargs))
            conv.append(layer.MaxPool2d(2, 2))


        self.conv_fc = nn.Sequential(
            *conv,

            layer.Flatten(),
            layer.Dropout(0.5),
            layer.Linear(channels * 4 * 4, 512),
            spiking_neuron(*args, **kwargs),

            layer.Dropout(0.5),
            layer.Linear(512, 110),
            spiking_neuron(*args, **kwargs),

            layer.VotingLayer(10)
        )

    def forward(self, x: torch.Tensor):
        return self.conv_fc(x)
```

Train

How to define the training method, loss function, and classification result are identical to previous tutorials, and we will not introduce them in this tutorial. We will only introduce the difference.

We use multi-step mode for faster training speed, and use *cupy* backend if *args.cupy*:

```
# spikingjelly.activation_based.examples.classify_dvsg

import torch
import sys
import torch.nn.functional as F
from torch.cuda import amp
from spikingjelly.activation_based import functional, surrogate, neuron
from spikingjelly.activation_based.model import parametric_lif_net
from spikingjelly.datasets.dvs128_gesture import DVS128Gesture
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import time
import os
import argparse
import datetime

def main():
    # ...
    net = parametric_lif_net.DVSGestureNet(channels=args.channels,
spiking_neuron=neuron.LIFNode, surrogate_function=surrogate.ATan(), detach_reset=True)

    functional.set_step_mode(net, 'm')
    if args.cupy:
        functional.set_backend(net, 'cupy', instance=neuron.LIFNode)
    # ...
```

Define the dataset:

```
# spikingjelly.activation_based.examples.classify_dvsg

def main():
    # ...
    train_set = DVS128Gesture(root=args.data_dir, train=True, data_type='frame',
frames_number=args.T, split_by='number')
    test_set = DVS128Gesture(root=args.data_dir, train=False, data_type='frame',
frames_number=args.T, split_by='number')
    # ...
```

Note that dimension 0 is always the batch dimension for data packed by `DataLoader`. So, the data we read from `DataLoader` has `shape = [N, T, C, H, W]`. We need to reshape the data to `shape = [T, N, C, H, W]` for the multi-step mode:

```
# spikingjelly.activation_based.examples.classify_dvsg

 def main():
    # ...
    for epoch in range(start_epoch, args.epochs):
        for frame, label in train_data_loader:
            optimizer.zero_grad()
            frame = frame.to(args.device)
            frame = frame.transpose(0, 1)  # [N, T, C, H, W] -> [T, N, C, H, W]
            # ...

        with torch.no_grad():
        for frame, label in test_data_loader:
            frame = frame.to(args.device)
            frame = frame.transpose(0, 1)  # [N, T, C, H, W] -> [T, N, C, H, W]
            # ...

    # ...
```

DVS Gesture has 11 classes:

```
# spikingjelly.activation_based.examples.classify_dvsg

def main():
    # ...
    label_onehot = F.one_hot(label, 11).float()
    # ...
```

`DVSGestureNet` does not output the pulse frequency, but the original output of `shape = [T, N, 11]`:

The networks in `spikingjelly.activation_based.model.parametric_lif_net` output spikes, rather than firing rates:

```
# spikingjelly.activation_based.model.parametric_lif_net

class DVSGestureNet(nn.Module):
    # ...
    def forward(self, x: torch.Tensor):
        return self.conv_fc(x)
```

Therefore, we need to average the output in the time-step dimension to get the firing rates, and then calculate the loss and accuracy by the firing rates:

```
# spikingjelly.activation_based.examples.classify_dvsg

def main():
    # ...
    out_fr = net(frame).mean(0)
    loss = F.mse_loss(out_fr, label_onehot)
    # ...
```

Train the network:

```
python -m spikingjelly.activation_based.examples.classify_dvsg -T 16 -device cuda:0 -b 16 -
epochs 64 -data-dir /datasets/DVSGesture/ -amp -cupy -opt adam -lr 0.001 -j 8
```

The outputs are:

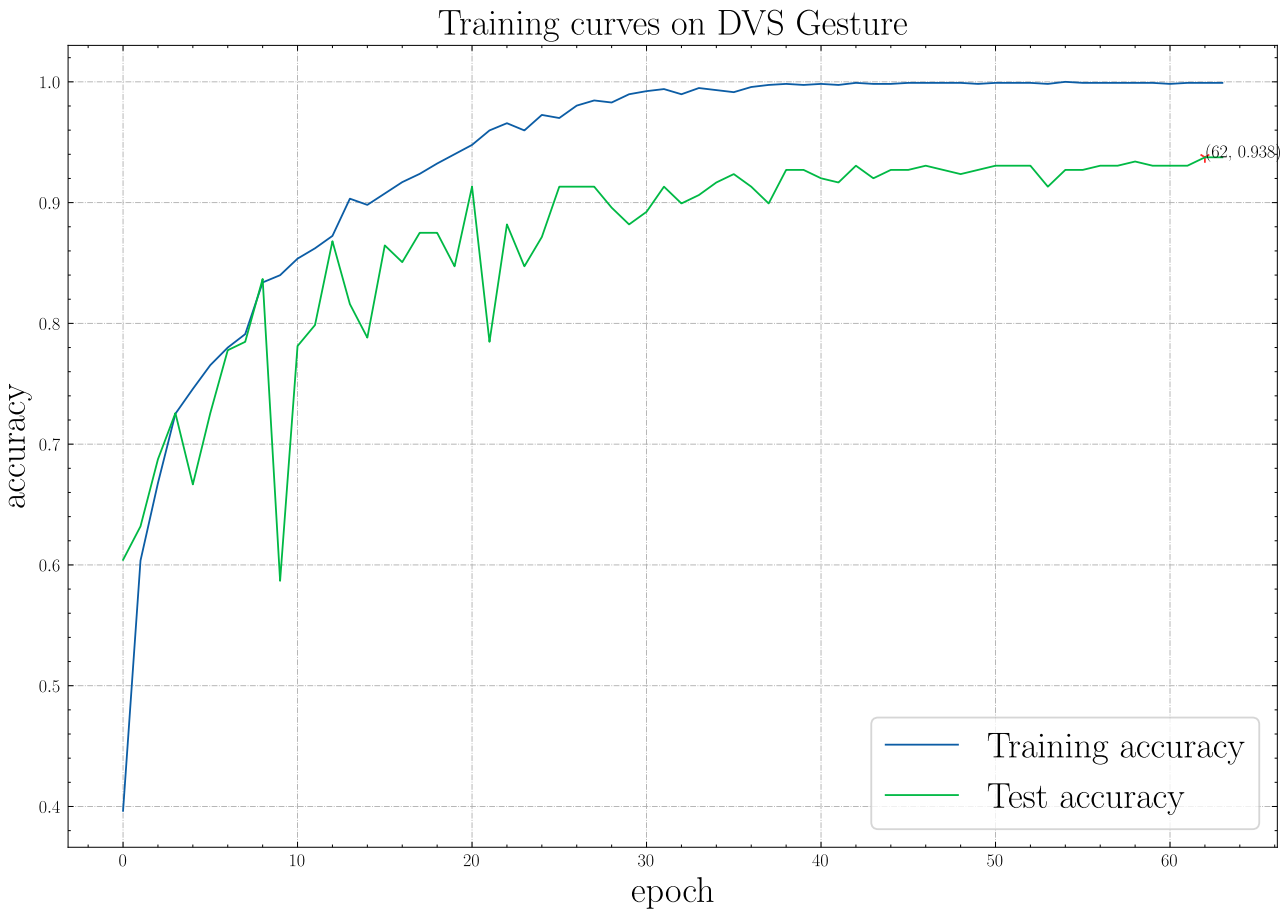
```
Namespace(T=16, device='cuda:0', b=16, epochs=64, j=8, data_dir='/datasets/DVSGesture/',
out_dir='./logs', resume=None, amp=True, cupy=True, opt='adam', momentum=0.9, lr=0.001,
channels=128)
DVSGestureNet(
(conv_fc): Sequential(
    (0): Conv2d(2, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False,
step_mode=m)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True,
step_mode=m)
    (2): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False,
step_mode=m)
    (4): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False,
step_mode=m)
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True,
step_mode=m)
    (6): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False,
step_mode=m)
    (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False,
step_mode=m)
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True,
step_mode=m)
    (10): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False,
step_mode=m)
    (12): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False,
step_mode=m)
    (13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True,
step_mode=m)
    (14): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False,
step_mode=m)
    (16): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False,
step_mode=m)
    (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True,
step_mode=m)
    (18): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False,
step_mode=m)
    (20): Flatten(start_dim=1, end_dim=-1, step_mode=m)
    (21): Dropout(p=0.5)
    (22): Linear(in_features=2048, out_features=512, bias=True)
    (23): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (24): Dropout(p=0.5)
```

latest

```
    (25): Linear(in_features=512, out_features=110, bias=True)
    (26): LIFNode(
    v_threshold=1.0, v_reset=0.0, detach_reset=True, step_mode=m, backend=cupy, tau=2.0
    (surrogate_function): ATan(alpha=2.0, spiking=True)
    )
    (27): VotingLayer(voting_size=10, step_mode=m)
)
)
The directory [/datasets/DVSGesture/frames_number_16_split_by_number] already exists.
The directory [/datasets/DVSGesture/frames_number_16_split_by_number] already exists.
Mkdir ./logs/T16_b16_adam_lr0.001_c128_amp_cupy.
Namespace(T=16, device='cuda:0', b=16, epochs=64, j=8, data_dir='/datasets/DVSGesture/',
out_dir='./logs', resume=None, amp=True, cupy=True, opt='adam', momentum=0.9, lr=0.001,
channels=128)
./logs/T16_b16_adam_lr0.001_c128_amp_cupy
epoch = 0, train_loss = 0.0666, train_acc = 0.3964, test_loss = 0.0514, test_acc = 0.6042,
max_test_acc = 0.6042
train speed = 92.7646 images/s, test speed = 115.2935 images/s
escape time = 2022-05-25 21:31:54

Namespace(T=16, device='cuda:0', b=16, epochs=64, j=8, data_dir='/datasets/DVSGesture/',
out_dir='./logs', resume=None, amp=True, cupy=True, opt='adam', momentum=0.9, lr=0.001,
channels=128)
./logs/T16_b16_adam_lr0.001_c128_amp_cupy
epoch = 1, train_loss = 0.0463, train_acc = 0.6036, test_loss = 0.0439, test_acc = 0.6319,
max_test_acc = 0.6319
train speed = 101.5938 images/s, test speed = 120.5184 images/s
escape time = 2022-05-25 21:30:48

...

Namespace(T=16, device='cuda:0', b=16, epochs=64, j=8, data_dir='/datasets/DVSGesture/',
out_dir='./logs', resume=None, amp=True, cupy=True, opt='adam', momentum=0.9, lr=0.001,
channels=128)
./logs/T16_b16_adam_lr0.001_c128_amp_cupy
epoch = 63, train_loss = 0.0011, train_acc = 0.9991, test_loss = 0.0103, test_acc = 0.9375,
max_test_acc = 0.9375
train speed = 100.4324 images/s, test speed = 121.0402 images/s
escape time = 2022-05-25 21:30:51
```

Finally, `max_test_acc = 0.9375` is achieved. Higher accuracy can be achieved if the hyper-parameters are carefully adjusted with more training epochs.

The following figure shows the accuracy curves during the training process:



[1] (1, 2) Fang, Wei, et al. “Incorporating learnable membrane time constant to enhance learning of spiking neural networks.” Proceedings of t