Ashwin Samuel (as4074) – Major in Computer Science
Anamika Lochab (al1522) – Major in Computer Science
Rishit Nagpal (rn443) – Major in Computer Science
Parth Khandelwal (pk684) – Major in Computer Science

Harnessing SNNs for Enhanced Radio Signal Classification

Ashwin Samuel, Anamika Lochab, Rishit Nagpal, Parth Khandelwal

05/03/23
525 Brain-Inspired Computing
Prof. Konstantinos Michmizos

Abstract

Radio communication is crucial in modern-day life, enabling efficient information exchange through various devices and services. With the increasing demand for communication services, effective spectrum management and signal classification are of paramount importance. In this project, we aim to re-implement a novel approach to modulation classification using Spiking Neural Networks (SNNs) in conjunction with the RadioML dataset. By leveraging the Spike Response Model (SRM) and Deep Continuous Local Learning (DCLL) for training, this project aims to get a lightweight, energy-efficient, and cost-effective solution for radio signal classification. Using this method, an accuracy of 99.02% is achieved between the classification of 2 different Modulations. This indicates an encouraging beginning in the pursuit of an effective, event-based approach for signal classification.

1. Introduction

The wireless communication landscape has undergone a remarkable transformation over the past few decades, with an ever-increasing number of connected devices and high data rate applications vying for limited spectral resources. As a result, the demand for intelligent and adaptive systems capable of discerning and adapting to various modulation schemes has grown tremendously. Modulation classification refers to the process of identifying and categorizing the modulation scheme employed by a received signal. Accurate classification of modulation schemes is particularly indispensable in cognitive radio networks and military communication systems, where the ability to detect and adapt to incumbent users is crucial for efficient spectrum sharing and secure communication.

Despite considerable progress in the field, modulation classification still presents numerous challenges and research opportunities. The increasing complexity of modern communication systems and the emergence of sophisticated modulation schemes have resulted in a more intricate classification landscape. Concurrently, rapid advancements in wireless technologies and the growing prevalence of machine learning and artificial intelligence techniques have introduced new dimensions to research in this area.

In recent years, Spiking Neural Networks (SNNs) have emerged as a promising approach to address the challenges associated with modulation classification as in [1] Ghasemzadeh et al. used deep belief networks (DBN) and spiking neural networks (SNN) for Modulation Recognition under a low signal-to-noise ratio. SNNs are the third generation of neural networks characterized by their ability to process and transmit information using temporal spike sequences, closely mimicking the functioning of biological neurons. SNNs naturally handle temporal dynamics in signals, making them well-suited for the time-varying nature of wireless communication channels. While SNNs typically achieve lower accuracy levels, they exhibit event-driven and sparse computation, improving energy efficiency – a critical factor in battery-powered devices and Internet of Things (IoT) applications. They can inherently tolerate noise and interference, enhancing their classification accuracy in real-world communication scenarios. The localized and asynchronous nature of SNNs enables easy parallelization and scalability, offering a viable solution for large-scale and complex classification problems.

This paper examines the role of SNNs in addressing the challenges associated with modulation classification in wireless communication. We explore key methodologies for modulation classification [7], including feature-based, likelihood-based, and deep learning-based

approaches, emphasizing the advantages of SNN-based techniques. In this project, we implement and optimize a Spiking Neural Network for modulation classification utilizing the RadioML dataset.

2. Background (or Theory)

Traditional modulation classification methods can be broadly categorized into likelihood-based and feature-based methods. While likelihood-based methods exhibit theoretical optimality, their computational requirements are substantial, as discussed by Xiao et al. in the challenges in [2]. Feature-based methods rely on manual feature extraction, which can be heavily dependent on expert experience. These methods are becoming less suitable for increasingly complex communication systems. Deep learning-based modulation recognition algorithms have shown promising results in recent years, outperforming traditional methods. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and other deep learning architectures have been employed for AMR, yielding high recognition accuracy.

Spiking Neural Networks (SNNs) are a class of artificial neural networks that attempt to model the behavior of biological neurons more accurately than traditional artificial neural networks. The Spike Response Model (SRM)[3] is a spiking neuron model class that simulates biological neurons' behavior. The SRM is characterized by its simplicity and computational efficiency, making it a suitable choice for simulating large-scale networks of spiking neurons. The SRM considers the effect of incoming spikes on the neuron's membrane potential, generating an output spike when the membrane potential reaches a certain threshold. But SNNs are difficult to train because of the issue of backpropagation.

The paper Synaptic Plasticity Dynamics for Deep Continuous Local Learning [4] provided an approach to resolve the backpropagation issue. Deep Continuous Local Learning (DCLL) is a learning framework specifically designed for training deep-spiking neural networks. DCLL focuses on local, biologically plausible learning rules, addressing the limitations of backpropagation-based learning in spiking networks. The main features of DCLL include continuous and local weight updates, which enable efficient learning and better biological plausibility compared to global weight updates in traditional learning methods.

SRM0 helps describe the neuron and synapse dynamics in discrete time. The membrane potential (U) of a neuron is governed by a set of equations involving the membrane potential (V), synaptic current (I), refractory state (R), and spike event (S). These equations can be rewritten regarding synaptic traces (P and Q), as shown in the equations.

$$U_i^l[t] = \sum_j W_{ij}^l P_j^l[t] + \rho R_i^l[t] + b_i^l$$

$$S_i^l[t] = 1 \; if \; U_i^l[t] \geq 0 \; else \; 0$$

$$Q_j^l[t + \Delta t] = \beta Q^l[t] + (1 - \beta) S_j^{l-1}[t]$$

$$P_j^l[t + \Delta t] = \alpha P_j^l[t] + (1 - \alpha) Q_j^l[t]$$

$$R_i^l[t + \Delta t] = \gamma R_i^l[t] - (1 - \gamma) S_i^l[t]$$

The global loss function is defined as the sum of layer-wise loss functions (L). To enforce locality, all non-local gradients are set to zero.

3. Experimental/Modeling Design

In this study, a spiking neural network (SNN) is implemented for the classification of radio signals using DCLL to train the model weights locally layer-wise, leveraging the unique temporal processing capabilities of SNNs for real-time classification.

RadioML

The RadioML dataset consists of 2.5 million radio signals formatted as I/Q samples over 1024 time steps each. Each signal is labeled with its modulation class (one of 24) and SNR (one of 26, with the minimum SNR being -20 dB and the maximum SNR being +30 dB).

Owing to the reliability of higher SNR signals, we do our classification for signals with SNR ranging from +6 dB to +30 dB. Our synaptic model is trained to classify between only two modulation schemes, 32PSK and 16APSK.

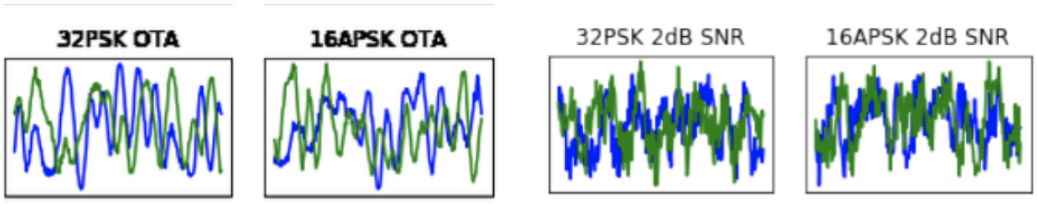

Figure 1: I/Q time-domain examples of 32PSK and 16 APSK

I/Q Sample Conversion to Spiking Input

To provide spiking input to the SNN, each I/Q sample is transformed (a tuple of two numbers representing the real and imaginary components of the signal at a time step) into an image by plotting the I/Q sample on the I/Q plane and discretizing the plane over the range [-1, 1] in each dimension.

Consider a signal with the I/Q samples for 3 timesteps as [(0.5,-1), (-0.5,0.5), (-0.5,1)] and image dimension of 16 by 16; we generate 3 images to be fed into our convolutional DCLL spiking network with the 16 by 16-pixel plane as 0 and only (12,1) pixel as 1 in the first image. Similarly, (4,12) and (4,16) will be 1. This allows the SNN to receive data points in the signal as events one after another as a spike train, enabling real-time class predictions.

We adopt the discrete-time Spike Response Model (SRM0) for simulating the neuron and synapse dynamics in our spiking neural network. The SRM0 is a simplified, computationally efficient model that captures the essential features of spiking neurons while facilitating the learning process. This model allows for a more manageable implementation of SNN, providing a balance between biological plausibility and computational tractability.

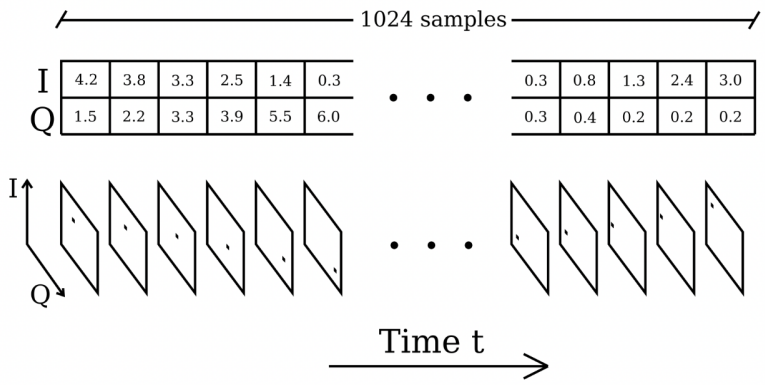

Figure 2: This is a visualization of image discretizations for I/Q samples

DCLL - an SNN Model with Plasticity Dynamics

A spiking neural network (SNN) is a type of neural network that simulates the communication of biological neurons through spikes over time. Unlike traditional neural networks, SNNs have a more complex neuron model and are well-suited to capturing temporal dynamics. However, SNNs are challenging to train due to the non-differentiability of spikes over time and the local computation constraint of biological neurons.

In this project, a novel method is utilized to solve the problem of non-differentiability, DCLL [4], for training an SNN. It solves the challenge of non-differentiability by utilizing auxiliary classifiers to provide local classification signals for each linear integrate and fire (LIF) neuron level in the SNN. LIF neurons emulate biological neurons and are defined by multiple state variables such as membrane potential and resetting state. This approach enables the successful training of SNNs, allowing for more effective temporal data processing in a neural network setting.

DCLL is an efficient way that uses layer-wise gradient calculation, which is only back-propagated locally instead of the entire network. Each layer is trained to do the classification task itself, but the output is forwarded to the next layer, as done normally, to an eventual fully connected layer which makes the final prediction. This ensures that each layer learns useful features for the task while only backpropagating local layer-wise gradients within the same layer and time step.

Parameter Tuning for Modulation Classification

In our efforts to train the SNN on RadioML data, a wide range of hyperparameter settings are explored. Some of the aspects we investigated included the resolution and range of discretized I/Q images, regularization weights, and various network architectural components such as the number of filters, pooling intensity, kernel sizes, padding types, convolutional and dense layers, and spiking or non-spiking dense layers. Additionally, various experiments were conducted with different constants for spiking neuron dynamics, activation functions, dropout probabilities, dropout layers, distinct sampling weights for classes and SNRs, and training with different SNRs and learning rates, including different learning rates for each layer.

Proposed Architecture

The proposed architecture for the SNN applied to the RadioML dataset draws inspiration from the VGG network architecture while making specific modifications to optimize performance for this dataset. Instead of the original 13 layers, the architecture employs 3 convolutional layers, which have proven effective in achieving high accuracy. Based on the results it was determined that a 16x16 resolution for the I/Q images provided sufficient detail for obtaining good accuracy.

Each layer learns 32 7x7 kernels with zero padding of length 3 on each side. The 3 convolutional layers are followed by a fully connected layer and a softmax activation to condense the intermediate results into class probabilities. To optimize the network, the Adam optimizer was employed with a smooth L1 loss as the loss function and a learning rate of 2.5e-10. This carefully designed architecture strikes a balance between performance and efficiency, meeting the specific needs of the signal classification problem at hand.

4. Results and Discussion

In this study, the layer-wise accuracy analysis of the model is performed for both training and testing datasets, made possible by the local layer-wise classification signals provided by DCLL. This examination enabled us to gain valuable insights into the performance of the neural network at each individual layer.

The results indicated that the accuracy of the two-layer model was inconsistent; however, we observed a substantial improvement in accuracy when progressing from the second to the third layer in both training and testing datasets. A total of 1024 runs in each epoch and depicted the accuracy as a function of an epoch, enabling us to monitor the model's progress over time.

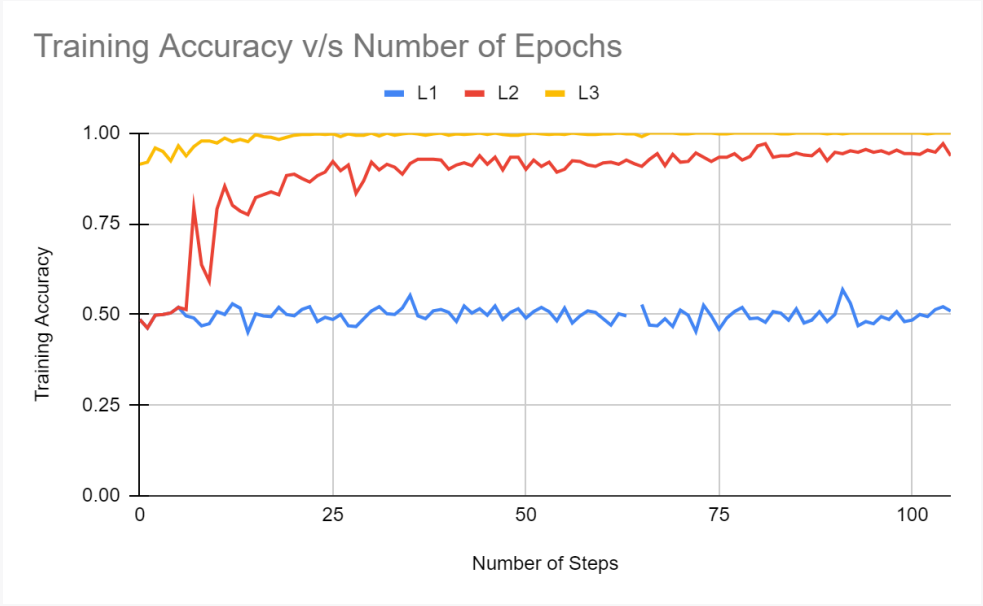

Figure 3: Graph for epoch vs. training accuracy

It was identified that learning started to plateau after a certain number of epochs, suggesting that further training would not lead to significant enhancements in accuracy. Since the training for the 2-class classification problem took 18 - 20 hours with a standard CPU, it is important to consider that a multi-class classification of all 24 modulation schemes in the dataset would have demanded longer training durations and might have necessitated hardware optimization or the employment of GPUs to make the process viable.

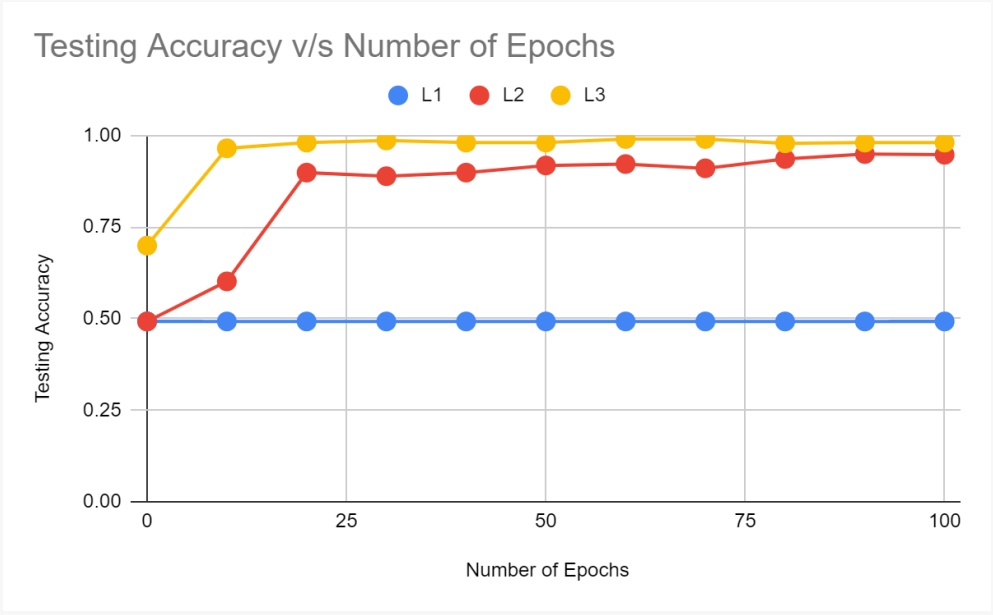

Figure 4: Graph for epoch vs. testing accuracy

The findings revealed that while most parameters did not substantially impact performance, the incorporation of a third layer led to a remarkable increase in accuracy. Following several experiments, we achieved an accuracy of 100% on the training data and 99.02% on unseen data for classifying signals between 32PSK and 16APSK modulation schemes. These outcomes underscore the efficacy of the proposed architecture and learning paradigm, DCLL, emphasizing its potential for practical implementation in signal processing and classification tasks.

5. Conclusions

In conclusion, this project demonstrates the potential of spiking neural networks (SNNs) as a viable and efficient approach to modulation classification in the context of limited frequency spectrum sharing. Employing SNNs using deep continuous local learning, shows that it is possible to achieve promising results on the RadioML dataset while maintaining a balance between accuracy and computational efficiency.

This provides the way for further optimization and exploration of SNN-based modulation classification methods. Future work could focus on enhancing the performance of the model and broadening it to classify more classes accurately and efficiently by refining the learning process and incorporating more advanced SNN architectures.

Acknowledgments

The authors express their gratitude to Professor Konstantinos Michmizos for providing the essential foundational material and guidance throughout the course of 525. His expertise and support have been invaluable in the development of this project.

References

[1] P. Ghasemzadeh, S. Banerjee, M. Hempel, and H. Sharif, "A Novel Deep Learning and Polar Transformation Framework for an Adaptive Automatic Modulation Classification," IEEE Transactions on Vehicular Technology, vol. 69, no. 11, pp. 13243-13258, Nov. 2020, doi: 10.1109/TVT.2020.3022394.

[2] Xiao, Wenshi, Zhongqiang Luo, and Qian Hu, "A Review of Research on Signal Modulation Recognition Based on Deep Learning" Electronics 11, no. 17: 2764, 2022 https://doi.org/10.3390/electronics11172764

[3] Gerstner, Wulfram, "Spike-response model" Scholarpedia 3 (2008): 1343.

[4] Kaiser Jacques, Mostafa Hesham, Neftci Emre, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)" Frontiers in Neuroscience 14, 2020, doi: 10.3389/fnins.2020.00424

[5] Simon Kaufmann, Owen Jow, "SNN for Modulation Classification" June 2020

[6] Tsakmalis, Anestis & Chatzinotas, Symeon & Ottersten, Björn, "Modulation and Coding Classification for Adaptive Power Control in 5G Cognitive Communications". IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC. 2014. 234-238. 10.1109/SPAWC.2014.6941505.

[7] T. Huynh-The et al., "Automatic Modulation Classification: A Deep Architecture Survey," IEEE Access, vol. 9, pp. 142950-142971, 2021, doi: 10.1109/ACCESS.2021.3120419.

Appendix

1. Reached 99.02% accuracy for the first time after 70 epochs. Test and training accuracy output after 70 epochs:

```
[TRAIN] Step 00070      Accuracy [0.466796875, 0.94140625, 1.0] -----> Timestamp = 2023-05-01 19:05:42.995776
[TEST]  Step 00070      Accuracy [0.4921875  0.91015625 0.99023438]
```

2. Main code train.py is as follows. For complete code, check this link

```
import torch
import numpy as np
import os
import argparse
import datetime

from dcll.pytorch_libdcll import device
from dcll.experiment_tools import mksavedir, save_source, annotate
from dcll.pytorch_utils import grad_parameters, named_grad_parameters, NetworkDumper,
 tonumpy
from networks import ConvNetwork, ReferenceConvNetwork, load_network_spec

from data.utils import to_one_hot
from data.load_radio_ml import get_radio_ml_loader as get_loader
from data.utils import iq2spiketrain as to_spike_train
if __name__ == '__main__':
   args = parse_args()
   torch.manual_seed(args.seed)
   np.random.seed(args.seed)

   get_loader_kwargs  = {}
   to_st_train_kwargs = {}
   to_st_test_kwargs  = {}

   #set no of iterations
   n_iters = args.n_iters
   n_iters_test = args.n_iters_test

   #image dimensions
   im_dims = (1, args.Q_resolution, args.I_resolution)

   #2 classes
   target_size = 2
```

```
# Set "get loader" kwargs
get_loader_kwargs['data_dir'] = args.radio_ml_data_dir
get_loader_kwargs['min_snr'] = args.min_snr
get_loader_kwargs['max_snr'] = args.max_snr
get_loader_kwargs['per_h5_frac'] = args.per_h5_frac
get_loader_kwargs['train_frac'] = args.train_frac

# Set "to spike train" kwargs
for to_st_kwargs in (to_st_train_kwargs, to_st_test_kwargs):
    to_st_kwargs['out_w'] = args.I_resolution
    to_st_kwargs['out_h'] = args.Q_resolution
    to_st_kwargs['min_I'] = args.I_bounds[0]
    to_st_kwargs['max_I'] = args.I_bounds[1]
    to_st_kwargs['min_Q'] = args.Q_bounds[0]
    to_st_kwargs['max_Q'] = args.Q_bounds[1]
to_st_train_kwargs['max_duration'] = n_iters
to_st_train_kwargs['gs_stdev'] = 0
to_st_test_kwargs['max_duration'] = n_iters_test
to_st_test_kwargs['gs_stdev'] = 0

# number of test samples: n_test * batch_size_test
n_test = np.ceil(float(args.n_test_samples) /
                 args.batch_size_test).astype(int)
n_tests_total = np.ceil(float(args.n_steps) /
                        args.n_test_interval).astype(int)

#set optimizer and loss function
opt = getattr(torch.optim, args.optim_type)
opt_param = {
    'betas': [0.0, args.beta],
    'weight_decay': 10.0,
}
loss = getattr(torch.nn, args.loss_type)

#load DCLL CNN based on parameters specified in  networks/radio_ml_conv.yaml
```

```
  burnin = args.burnin
  convs = load_network_spec(args.network_spec)
  net = ConvNetwork(args, im_dims, args.batch_size, convs, target_size,
                                          act=torch.nn.Sigmoid(),  loss=loss,  opt=opt,
opt_param=opt_param,
                      learning_rates=args.learning_rates, burnin=burnin)

  #network init before training starts
  net = net.to(device)
  net.reset(True)
  acc_test = np.empty([n_tests_total, n_test, len(net.dcll_slices)])

  #extract readable numpy train data from hdf5 files in dataset
  train_data = get_loader(args.batch_size, train=True, **get_loader_kwargs)
  gen_train = iter(train_data)
  gen_test = iter(get_loader(args.batch_size_test, train=False, **get_loader_kwargs))

  all_test_data = [next(gen_test) for i in range(n_test)]
  all_test_data = [(samples, to_one_hot(labels, target_size))
                   for (samples, labels) in all_test_data]

  for step in range(args.n_steps):
      if ((step + 1) % 1000) == 0:
          for i in range(len(net.dcll_slices)):
              net.dcll_slices[i].optimizer.param_groups[-1]['lr'] /= 2
          net.dcll_slices[-1].optimizer2.param_groups[-1]['lr'] /= 2
          print('Adjusting learning rates')

      #extract data 1 batch_size at a time (default = 512)
      try:
          input, labels = next(gen_train)
      except StopIteration:
          gen_train = iter(train_data)
          input, labels = next(gen_train)

      labels = to_one_hot(labels, target_size)
```

```
      #no of images, out of the 1024 timesteps per signal (1 timestep I/Q value is 1
image), sent into the network for training
      n_iters_sampled = n_iters
      to_st_train_kwargs['max_duration'] = n_iters_sampled

      #convert image to data to spike train
      input_spikes, labels_spikes = to_spike_train(input, labels,
                                                   **to_st_train_kwargs)
      input_spikes = torch.Tensor(input_spikes).to(device)
      labels_spikes = torch.Tensor(labels_spikes).to(device)

      #n_iters images generated from each of the 512 signals is sent into the network
for training
      net.reset()
      net.train()
      for sim_iteration in range(n_iters_sampled):
          net.learn(x=input_spikes[sim_iteration],
                      labels=labels_spikes[sim_iteration])

      #Training accuracy acheived for each dcll slice / each layer
      acc_train = net.accuracy(labels_spikes)
      step_str = str(step).zfill(5)
       print('[TRAIN] Step {} \t Accuracy {} -----> Timestamp = {}'.format(step_str,
acc_train,datetime.datetime.now()))

      # Test after every n_test_interval (default=10)
      if (step % args.n_test_interval) == 0:
          test_idx = step // args.n_test_interval
          for i, test_data in enumerate(all_test_data):

              #convert test data to spike train
              test_input, test_labels = to_spike_train(*test_data,
                                                       **to_st_test_kwargs)
              try:
                  test_input = torch.Tensor(test_input).to(device)
```

```
            except RuntimeError as e:
                print('Exception: ' + str(e) +
                                  '. Try to decrease your batch_size_test with the
--batch_size_test argument.')
                raise
            test_labels = torch.Tensor(test_labels).to(device)


              #send in a vector of batch size x no of images per signal and run for
all images per signal
              #prediction is the class predicted max no of times in the 1024 images
for each signal
            net.reset()
            net.eval()


            for sim_iteration in range(n_iters_test):
                net.test(x=test_input[sim_iteration])


            acc_test[test_idx, i, :] = net.accuracy(test_labels)


        #print the accuracy
        acc = np.mean(acc_test[test_idx], axis=0)
        step_str = str(step).zfill(5)
        print('[TEST]  Step {} \t Accuracy {}'.format(step_str, acc))
```