AJ - Masters in CS – Concentration – General CS
SK - Masters in CS – Concentration – Machine Learning
SG - Masters in CS – Concentration – General CS
UW - Masters in CS – Concentration – General CS

TRAFFIC SIGN RECOGNITION USING SPIKING NEURAL NETWORK

Abhishek Jain
Shreyas Kulkarni
Swathi Gopal
Udit Wadhera

05/07/22

525 Brain Inspired Computing
Prof. Konstantinos Michmizos

Abstract

Autonomous driving cars is one of the major fields that is developing rapidly. With the keen expectations from automated vehicles, research on this topic enabled a practical application of fully automated vehicles. Abiding to traffic signs is the first rule for easy transportation and hence recognition of traffic signs becomes extremely critical. Considering the real-world situations like light interference, weather conditions and other potential hindrances, it gets harder to detect and recognize a traffic sign. Here a 3rd generation artificial neural network, Spiking Neural Network, is used to recognize and classify the traffic signs. German Traffic Sign Dataset is used and a binary classification of stop and yield signs is implemented using SNN. An accuracy of 97.40% for training and 98.5% testing is achieved. This approach can later be extended to discern different classes of traffic signs. This method is energy and time efficient compared to other traditional neural networks.

1. Introduction

Machine Learning applications are now widely used in a variety of science and research domains, allowing them to improve the outcomes of a wide range of automated and non-automated tasks, as well as other application areas. One of the primary fields that uses machine learning techniques is to advance automated tasks in autonomous driving cars. In this present era of technology, many multinational companies are focused on these automatic driving cars.

Traffic signs are an integral element of every driver's daily routine. Several essential environmental details can be accessed on traffic signs that can assist drivers in learning about upcoming road changes and driving restrictions. It is very important for an automated car to detect and analyze the traffic sign in no time. An increasing number of researchers have focused on the difficulties of traffic sign recognition utilizing computer vision and machine learning approaches. However, there are many difficulties like 1) poor weather conditions, 2) poor image quality 3) illumination condition 4) deterioration of signs.

Many techniques have been implemented to yield these above tasks. Basically a traffic recognition algorithm contains two steps: traffic sign detection and traffic sign recognition. Traffic sign detection is done based on color based or shape based segmentation techniques. In traffic sign recognition different machine learning approaches like SVM, ADABOOST, Neural Networks are used.

Traditional Machine learning algorithms need extensive time-consuming manual work to discern the traffic signs. The focus is on Traffic Sign Recognition using SNN. Spiking neural networks (SNNs) are based on biological information processing, in which dense and asynchronous binary signals are transmitted and processed parallely. Low power consumption, quick inference, and event-driven information processing makes SNNs better for implementing deep neural networks.

In the proposed algorithm an image is fed to the neural network. This image is filtered, reconstructed and converted to event images. Training is then performed on these images.

2. Background (or Theory)

A wide range of papers were examined while understanding the area of Traffic Sign Detection and Classification. An article titled "Road Traffic Sign Detection and Classification" [4] published in 1997 experimented numerous solutions to the discussed problem. The detection of traffic signs was achieved in stages. First, Color Thresholding was applied to the original image in a modified HSI color space. The second step was to detect corners with a convolutional mask. For various traffic sign shapes, different masks were created. Finally, after all corners have been detected, the center of mass for each traffic sign is calculated. A combination of neural networks and traditional image processing methods was used to classify traffic signs. After detecting a traffic sign, the nearest neighbor method was used to resize it to a 30 x 30 pixel image. The image was then fed into one of two neural networks based on the shape of the sign. One neural network was for triangular signs, while the other was for circular signs. The sign detection algorithm took an average of 220 ms to process a 256x256 image and the neural network took 1.2s to process using a PC486 processor. But testing was conducted on a very small dataset, and no detection and classification accuracy results were provided.

With the evolution of processors and graphic cards more efficient algorithms were developed. So the use of machine learning for image classification tasks started around 2015-2016. Currently machine learning uses the 2nd generation of Artificial Neural networks. These are fully connected neural networks that take continuous input value and generate continuous output. The traffic sign recognition using CNN/ANN algorithm on German Traffic Sign Data Set has an accuracy of 99% (accuracy for 43 classes) [5]. Although they are very successful, the processing and energy requirements while training and deploying these networks is unsustainable. This makes it necessary for a new approach [6].

The 3rd generation, spiking neural networks bridges the gap between biologically realistic neurons and machine learning. Spiking Neural networks use spikes that are discrete events happening at various times. The spike occurrence is described by a differential equation which is based on the neuron's membrane potential.

Figure1. illustrates two spiking neuron pre and post connected by a synapse of weight w. The membrane potential of neuron is an essential factor

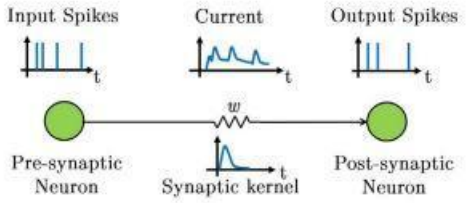

Figure 1: Illustrating Connection between pre and postsynaptic neuron

This is modeled by Leaky Integrate-Fire Neuron model(LIF). It is shown by equation1 where I(t) represents the excitation current, C and $g_{L,}$ represents membrane capacitance and leak conductance. Once V(t) crosses the threshold value it is reset to its resting voltage.

$$C\,(dV\,/(t)\,dt) = -g_L(V\,(t) - E_L) + I(t)$$

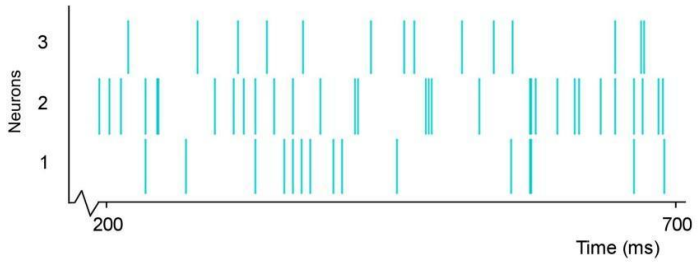

Figure 2: Spike train of 3 neurons

In real world applications the data is spatio-temporal. Spike trains help in processing these types of data. The spatial component means that neurons are only connected to neurons in their immediate vicinity, so they process inputs separately. Temporal component means that the spikes occur over time so the loss in binary encoding is gained backed in temporal information.

3. Experimental/Modeling Design

1)Network Architecture:

The SNN model used here is a 3 layered feed forward network where each neuron is connected to all the neurons in the next layer. The network has 784 neurons in the input layer and 2 neurons in the output layer for classifying two classes. All the neurons from the input layer are connected to the output layer through the weighted hidden(synaptic) layer. The spike train inputs are fed to the input layer. The membrane potential gets updated after each time step based on learning rule and weights. Early firing neurons in the output layer inhibits other neurons from firing.

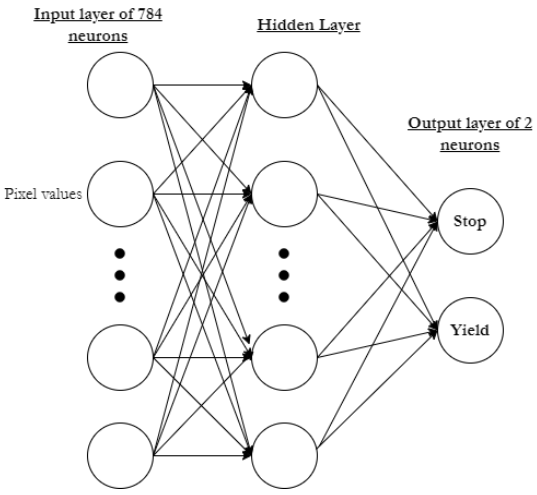

Figure 3: Spiking Neural Network

1.1) German Traffic Sign Dataset:

The proposed algorithm is tested on the German Traffic Sign Dataset. This data set has 43 classes with more than 50,000 images of size 32*32 with different backgrounds. Here, the implementation is restricted to two classes: “Stop Sign” and “Yield Sign”. The total number of images that is used in this implementation is 5549.

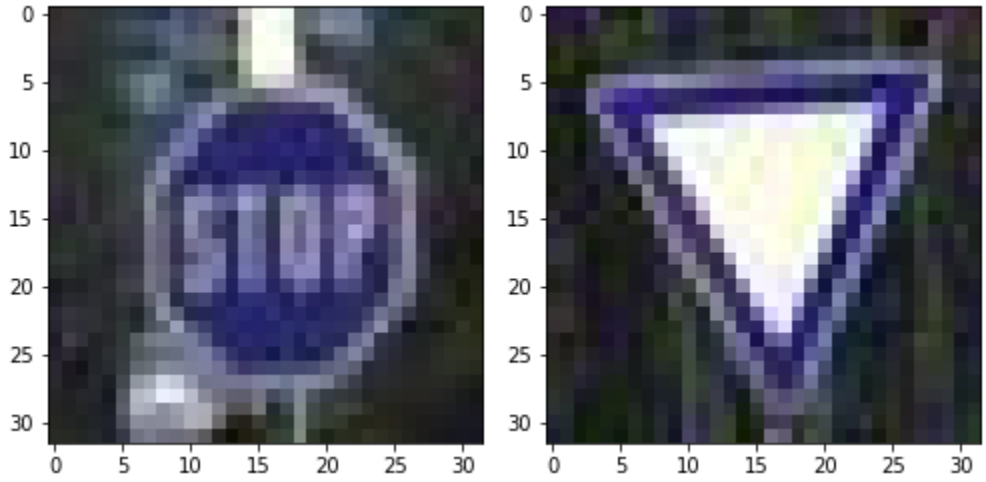

Figure 4: Image of Stop sign and Yield sign from dataset

1.2) Rate Encoding:

The raw image is fed to the network. Image filtering is performed to extract the region of focus. Image filtering helps in improving the analysis by suppressing other frequencies and extracting the part that's important for the classification. Here, we extract the white portion of the image. The parameters that were used for this are:[0,0,138] as lower bound and [255,117,255] as upper bound. The image is then reconstructed to 28*28. The input image is a continuous valued function. This is converted to spikes by a method called rate encoding. The images are encoded in such a way that the frequency of spike is equal to the pixel intensity. Rate encoding is also defined as mean firing rate. Below figure shows the encoded image using rate encoding.

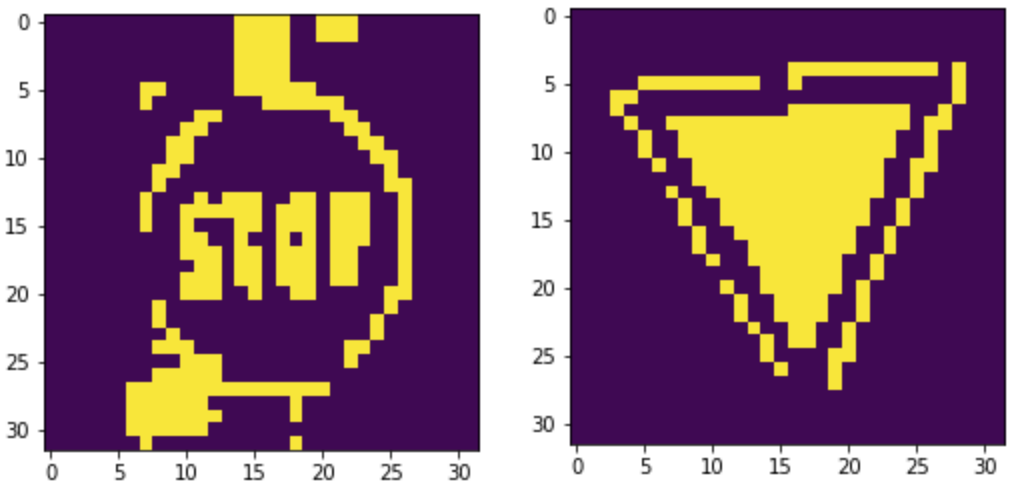

Figure 5: Rate Encoded Image of Stop sign and Yield sign

2)Stochastic Gradient Descent Using Backpropagation:

Describing the network architecture, it is clear that each neuron in the hidden layer is parameterized with weights. Now the goal is to calculate the value of weights such that it minimizes the loss in the network output. Given the image and the label, the network is trained to learn its weight. H is a nonlinear function of weighted inputs and it is represented as

$$H=\sigma(w,x),$$

where w represents weight and x represents the input. Now the loss is computed in the network. Loss is computed as mean squared error between the predicted and true label.

$$L=1/2*(Y_{predicted}-Y)^2$$

Where $Y_{predicted}$ is the predicted output and Y is the true label.

w_{ij} is the weight of a connection from output of i^{th} node to input of n^{th} node. X being the input to the node and Y being the output, activation function is defined as

$$Y=F(X)$$

The primary aim in training is to minimize the loss. Here Stochastic Gradient descent is used, the weights w is moved in the negative direction of loss function L according to

$$w=w-\eta\partial L/\partial w,$$

where η is the learning rate.

The Backpropagation algorithm is used here to compute the gradient of loss function. This algorithm uses chain rule to calculate the partial derivatives. The algorithm is broken into two parts:

i)Forward Pass: Here the input is propagated in the forward direction and computes the pre-activation and activation function for the output layer. This stage is also called prediction.

ii)Backward Pass: The error generated(derivative) is passed backward through the same nodes and connections to the lower layer. This stage is called as training

The input is propagated from input to output layer via forward pass, the gradient/loss of this is calculated by backward pass from the output layer to input layer and then the weights are updated.

A problem that was faced here is that the spikes are not differentiable function. So a rectangular function was used to approximate this.

The images in the data set is divided into two parts: 5149 for training and 200 for testing. The neural network was trained for 30 epochs.

3) Testing the network:

A completely trained network with learned weights is now available. With this, the network output is computed for the Training data set and it is decoded to predict the class. As the class is predicted, it is compared with its true label and accuracy of the model is computed.

4. Results and Discussion

A SNN classifier that classifies two different classes : Stop Sign and yield sign has been implemented. Performing training on 5149 images from the German traffic sign data set, we achieved an accuracy of 97.40%.

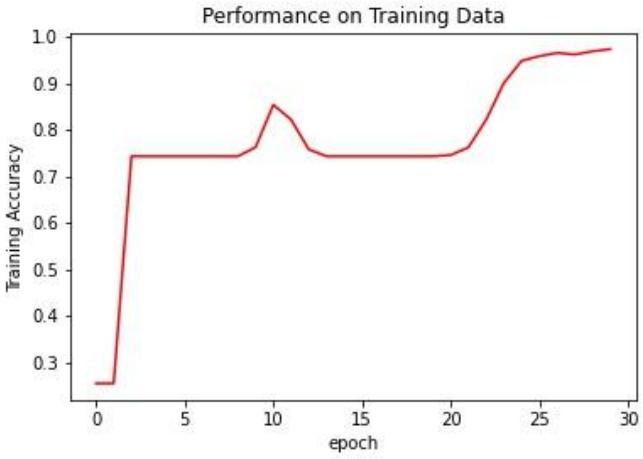

Figure 6: Graph depicting training accuracy

The accuracy increases till 4 epochs and for the next 15 to 20 epochs it stays constant. Until this point the model trains itself with the first class and then after the accuracy starts increasing again. This happens as initially the model is trained only for 1 class . A final accuracy of about 98.5% is achieved in the testing phase.

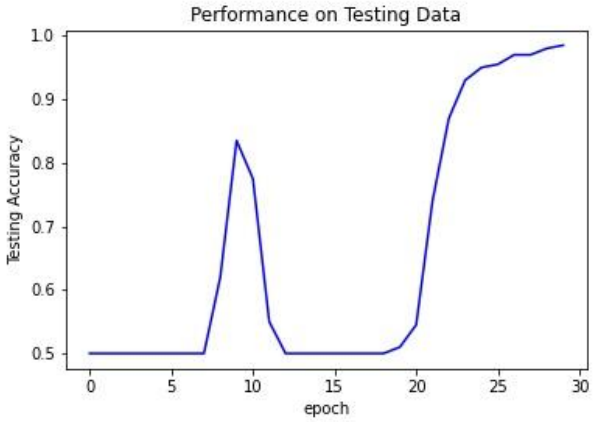

Figure 6: Graph depicting Testing accuracy

For 2 classes and 200 images for testing, above-mentioned accuracy is achieved but this varies as the number of classes and images are increased.

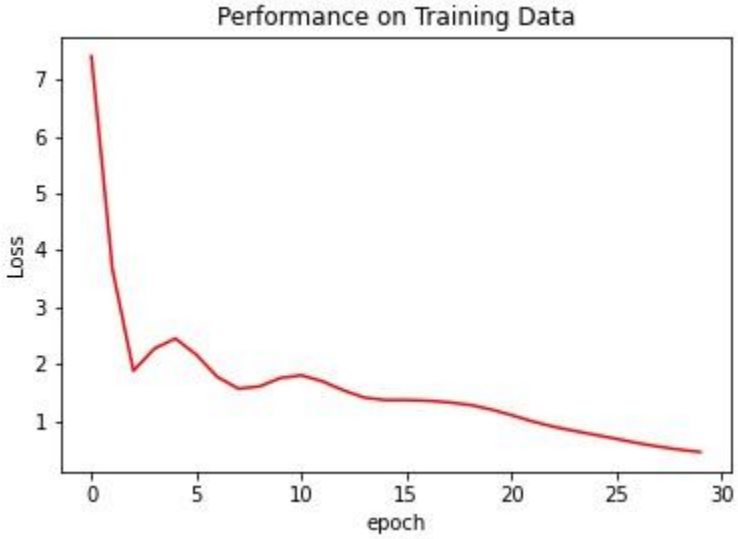

Figure 6: Graph depicting training loss

One of the better accuracy of 99.95% was achieved using the GHT+CNN method for 9877 images.[11]. The method employing ConvNet for both localizing and categorizing traffic signs is one of the most efficient methods for TSR using GTSDB and GTSRB. The authors show that when detecting a sign and classifying it, they can achieve precision of 99.89 percent and 99.55 percent, respectively[12]. Compared to these methods implementation using snn is energy efficient and it has quick inference as well as same or better accuracy.

During the implementation of the model, the difficulties were faced while generating the event image. Initially the raw image was fed directly to the model and hence the event images generated were distorted. This was solved by filtering the image to extract the white part of the image.

5. Conclusion

A method for traffic sign recognition is discussed in this paper. A binary classifier to classify stop and yield signs using a spiking neural network is implemented. The algorithm uses the SNN Torch library. Algorithms mainly consist of input encoding, training phase and testing phase. The image obtained is resized and is converted to spike trains using rate encoding. Later the weights of the network are trained using a backpropagation method in stochastic gradient descent. Once the network has learned its weights, the output label is predicted for the given input. This algorithm was experimented on the German Traffic Sign Data Set. For two classes, a training accuracy of 97.40% and testing accuracy of 98.5%is achieved.

The observed accuracy can be improved by better image pre-processing techniques and better filtering. The qualities of images are poor based on the practical aspects. Implementation of image preprocessing techniques before the classification task will lead to higher accuracy.

The realm of traffic signs is very broad. This can be extended to different classes in traffic signs. The German data set has 43 classes; this approach can be extended to include all the classes. Here due to time constraints only 2 classes were selected and classifiers were implemented. As SNN proves to be computationally efficient, energy efficient, has low power consumption and has quick inference, hence deploying a fully built Traffic sign recognition network in automated cars will result in better performances

Acknowledgments

The authors sincerely acknowledge Prof. Konstantinos Michmizos for the guidelines provided throughout the project and course work. We also extend our gratitude to Neelesh for providing continuous help and support.

References

[1] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. GomezMoreno, and F. Lopez-Ferreras, "Road-sign detection and recognition based on support vector machines," Ieee Transactions on Intelligent Transportation Systems, vol. 8, pp. 264-278, Jun 2007

[2] M. Garcia-Garrido, M. Sotelo, and E. Martin-Gorostiza, "Fast Road Sign Detection Using Hough Transform for Assisted Driving of Road Vehicles," in Computer Aided Systems Theory - EUROCAST 2005, 2005, pp. 543-548.

[3] L. D. Lopez and O. Fuentes, "Color-based road sign detection and tracking," in Image Analysis and Recognition, Proceedings. vol. 4633, M. Kamel and A. Campilho, Eds., ed Berlin: Springer-Verlag Berlin, 2007, pp. 1138-1147

[4] Davidson, S., & Furber, S. B. (2021). Comparison of Artificial and Spiking Neural Networks on Digital Hardware. *Frontiers in Neuroscience*, *15*. https://doi.org/10.3389/fnins.2021.651141

[5] German Traffic Sign Benchmark. Available at: http://benchmark.ini.rub.de/ [7] J. Redmon, S. Divvala, R. Girshick

[7] Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*(NOV). https://doi.org/10.3389/fnins.2016.00508

[8] Kulkarni, S. R., & Rajendran, B. (2018). Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. *Neural Networks*, *103*. https://doi.org/10.1016/j.neunet.2018.03.019

[9] Fan, Y., & Zhang, W. (2016). Traffic sign detection and classification for Advanced Driver Assistant Systems. *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015*. https://doi.org/10.1109/FSKD.2015.7382137

[10]Ciuntu, V., & Ferdowsi, H. (2020). Real-Time Traffic Sign Detection and Classification Using Machine Learning and Optical Character Recognition. *2020 IEEE International Conference on Electro Information Technology (EIT)*. doi:10.1109/eit48999.2020.9208309

[10]W. H. D. Fernando and S. Sotheeswaran, "Automatic road traffic signs detection and recognition using 'You Only Look Once' version 4 (YOLOv4)," 2021 International Research Conference on Smart Computing and Systems Engineering (SCSE), 2021, pp. 38-43, doi: 10.1109/SCSE53661.2021.9568285.

[11]Shustanov, A., & Yakimov, P. (2017). CNN Design for Real-Time Traffic Sign Recognition. In *Procedia Engineering* (Vol. 201, pp. 718–725). Elsevier Ltd.

[12] H. Aghdam, E. Heravi, D. Puig, A practical approach for detection and classification of traffic signs using Convolutional Neural Networks, Robotics and Autonomous Systems. 84 (2016) 97-112.

Appendix

```
#Snn torch Installation
!pip install snntorch --quiet
#Importing Libraries
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from PIL import Image
import os
import tensorflow as tf
import torch, torch.nn as nn
import snntorch as snn
import glob
from skimage import color
from skimage import io
```

```
from google.colab import drive
#Data Loading
drive.mount('/content/drive')
device = torch.device("cuda") if torch.cuda.is_available() else
 torch.device("cpu")
#Data Loading
image_data_train = []
image_labels_train = []
!ls drive/MyDrive/MyTrafficWorkingDataSet/Data
data_dir = r'drive/MyDrive/MyTrafficWorkingDataSet/Data'
for i in range(2):
    path =os.path.join(data_dir,str(i+1))
    print('Patth: '+path)
    for img in os.listdir(path):
        try:

            image = cv2.imread(os.path.join(path,img),0)
            image_fromarray = Image.fromarray(image, 'L')

            resize_image = image_fromarray.resize((28,28))
            image_data_train.append(np.array(resize_image))
            image_labels_train.append(i)
        except:
            print("Error in " + img)

#Reshaping the data
image_data_train = np.array(image_data_train)
image_data_train = np.expand_dims(image_data_train, axis=-1)
image_labels_train = np.array(image_labels_train)

print(image_data_train.shape, image_labels_train.shape)

image_data_train = image_data_train.astype(np.float32)
print(image_data_train.dtype)
```

```
#Changing the axis
print(image_data_train.shape, image_labels_train.shape)
rearranged_arr_train = np.moveaxis(image_data_train, [3,1,2],[1,2,3])
image_data_train = rearranged_arr_train
print(rearranged_arr_train.shape, image_labels_train[5000])
 if torch.cuda.is_available():
      device = 'cuda'
      print('cuda')
 else:
      device = 'cpu'
      print('cpu')

#  Initialize Network

from snntorch import surrogate
beta = 0.9
spike_grad = surrogate.fast_sigmoid()


net = nn.Sequential(nn.Conv2d(1, 8, 5),
                    nn.MaxPool2d(2),
                              snn.Leaky(beta=beta,  spike_grad=spike_grad,
 init_hidden=True),
                    nn.Conv2d(8, 16, 5),
                    nn.MaxPool2d(2),
                              snn.Leaky(beta=beta,  spike_grad=spike_grad,
 init_hidden=True),
                    nn.Flatten(),
                    nn.Linear(16*4*4, 2),
                              snn.Leaky(beta=beta,  spike_grad=spike_grad,
 init_hidden=True, output=True)
                    ).to(device)
# forward pass
from snntorch import utils
```

```
def forward_pass(net, data, num_steps):
  spk_rec = []
  utils.reset(net)

  for step in range(num_steps):
      spk_out, mem_out = net(data)
      spk_rec.append(spk_out)

  return torch.stack(spk_rec)
#optimizer and loss
import snntorch.functional as SF

optimizer   =   torch.optim.Adam(net.parameters(),   lr=2e-3,   betas=(0.9,
 0.999))
loss_fn = SF.mse_count_loss(correct_rate=0.8, incorrect_rate=0.2)
i = 0
#Test Image load
image_data_test = []
image_labels_test = []
!ls drive/MyDrive/MyTrafficWorkingTest/Data
data_dir = r'drive/MyDrive/MyTrafficWorkingTest/Data'
for i in range(2):
    path =os.path.join(data_dir,str(i+1))
    print('Patth: '+path)
    for img in os.listdir(path):
        try:

            image = cv2.imread(os.path.join(path,img),0)
            image_fromarray = Image.fromarray(image, 'L')

            resize_image = image_fromarray.resize((28,28))
            image_data_test.append(np.array(resize_image))
            image_labels_test.append(i)
```

```
        except:
            print("Error in " + img)

# Changing the list to numpy array
image_data_test = np.array(image_data_test)
image_data_test = np.expand_dims(image_data_test, axis=-1)
image_labels_test = np.array(image_labels_test)
print(image_data_test.shape, image_labels_test.shape)
image_data_test = image_data_test.astype(np.float32)
print(image_data_test.dtype)

#rearranging the test code
rearranged_arr_test = np.moveaxis(image_data_test, [3,1,2],[1,2,3])
image_data_test = rearranged_arr_test
print(rearranged_arr_test.shape, image_labels_test.shape)
# training and testing code
num_epochs = 30
num_steps = 25  # run for 25 time steps

loss_hist_train = []
acc_hist_train = []
acc_hist_test = []

for epoch in range(num_epochs):
    data = image_data_train
    targets = image_labels_train
    data = torch.FloatTensor(data)
    targets = torch.FloatTensor(targets)

    data = data.to(device)
    targets = targets.to(device)

    net.train()
    spk_rec = forward_pass(net, data, num_steps)
```

```
        loss_val = loss_fn(spk_rec, targets)

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
        loss_hist_train.append(loss_val.item())

        if i % 25 == 0:
          print(f"Epoch {epoch}, \nTrain Loss: {loss_val.item():.2f}")

        acc = SF.accuracy_rate(spk_rec, targets)
        acc_hist_train.append(acc)
        print(f"Accuracy: {acc * 100:.2f}%\n")
        total = 0
        acc = 0
        net.eval()
        data_test = image_data_test
        targets_test = image_labels_test
        data_test = torch.FloatTensor(data_test)
        targets_test = torch.FloatTensor(targets_test)

        data_test = data_test.to(device)
        targets_test = targets_test.to(device)

        spk_rec = forward_pass(net, data_test, num_steps)
        acc += SF.accuracy_rate(spk_rec, targets_test) * spk_rec.size(1)
        total += spk_rec.size(1)
        acc_hist_test.append(acc/total)
        print("Test Accuracy ", acc/total)
    #printing the values
    print("Training Accuracy ", acc_hist_train)
    print("Testing Accuracy ", acc_hist_test)
    print("Loss ", loss_hist_train)
    Training_Accuracy = acc_hist_train
```

```
Testing_Accuracy = acc_hist_test
Loss = loss_hist_train
len(Loss)
X = []
for i in range(30):
  X.append(i)
#graph plotting
plt.plot(X, Training_Accuracy, color='red')
plt.xlabel('epoch')
plt.ylabel('Training Accuracy')
plt.title('Performance on Training Data')
plt.savefig('Train.jpeg')
plt.show()
plt.plot(X, Testing_Accuracy, color='blue')
plt.xlabel('epoch')
plt.ylabel('Testing Accuracy')
plt.title('Performance on Testing Data')
plt.savefig('Test.jpeg')
plt.show()
plt.plot(X, Loss, color='red')
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.savefig('loss.jpeg')
plt.show()
```