Image Classification using SNN architecture with Brain Learning Algorithm

Session Monsoon 2020

Submitted By

Pranika Kakkar

(1710110253)

Rishika Chaudhary

(1710110276)

Under Supervision

of

Dr. Sonal Singhal

(Assoc. Prof., Electrical Engineering Department)

Department Of Electrical Engineering
School Of Engineering
Shiv Nadar University

(Spring 2020)

Candidate Declaration

I hereby declare that the thesis entitled “Image Classification using SNN architecture with Brain Learning Algorithm” submitted for the B.Tech Degree program. This thesis has been written in my own words. I have adequately cited and referenced the original sources.

(Pranika Kakkar)

(1710110253)

(Rishika Chaudhary)

(1710110276)

Date: November 18, 2020

CERTIFICATE

It is certified that the work contained in the project report titled “Image Classification using SNN architecture with Brain Learning Algorithm,” by “Pranika Kakkar, Rishika Chaudhary,” has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

Dr. Sonal Singhal
Electrical Engineering Department
School of Engineering
Shiv Nadar University
November,2020

ABSTRACT

Neuromorphic computing research emulates the neural structure of the human brain. The computational building blocks within neuromorphic computing systems are logically analogous to neurons. Therefore, in recent years there is a burgeoning interest in how spiking neural networks can be utilized in order to perform complex computations or solve various tasks like pattern recognition, speech recognition, image classification etc. Training strategy for SNNs can be broadly categorized into unsupervised and supervised algorithms. Unsupervised algorithms discover the characteristics and underlying structures of input patterns without using the corresponding output labels. Spike-Timing-Dependent-Plasticity (STDP) is a bio-plausible unsupervised learning mechanism that instantaneously manipulates the synaptic weights based on the temporal correlations between pre- and postsynaptic spike timings.In this work a simplified neuron model with a brain-learning algorithm will be used to implement SNN for image classification tasks by using python. Python code has been implemented for the network elements which are neuron, synapse receptive field and spike train. After employing the brain learning algorithm, we will reconstruct our input image back with minimum errors using the Lateral Inhibition process for optimization.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

Neuromorphic computing has been been around for decades in the field of artificial intelligence. Particularly in the field of image classification, the data is now being encoded into spike trains and seems ideal for benchmarking of neuromorphic learning algorithms [1]. The concept that neural information is encoded in the firing rate of neurons has been the dominant paradigm in neurobiology for many years. This paradigm has also been adopted by the theory of artificial neural networks. Recent physiological experiments demonstrate, however, that in many parts of the nervous system, neural code is founded on the timing of individual action potentials. This finding has given rise to the emergence of a new class of neural models, called spiking neural networks.

1.1 Spiking Neural Network

The spiking neural networks (SNNs) are artificial neural networks that imitate natural neural networks. They include the concept of time along with neuronal and synaptic state. The neurons in the SNN do not fire at every propagation cycle, instead fire only when the membrane potential reaches a specific threshold value. When a neuron is fired it generate signals that travel to other neurons which, in turn change their potentials proportionally with the signal (it can either increase or decrease).

The SNN model used in this work is the feed-forward network, each neuron is connected to all the neurons in the next layer by a weighted connection, which means that the output signal of a neuron has a different weighted potential contribution . Input neurons require spike trains and input signals (stimuli) need to be encoded into spikes (typically, spike trains) to further feed the SNN.
The figure 1.1 delineates the basic model of a spiking neural network where x1, x2, x3, x4 are the inputs for the model. After several incoming spikes, the membrane potential surpasses threshold and neuron fires a postsynaptic spike.

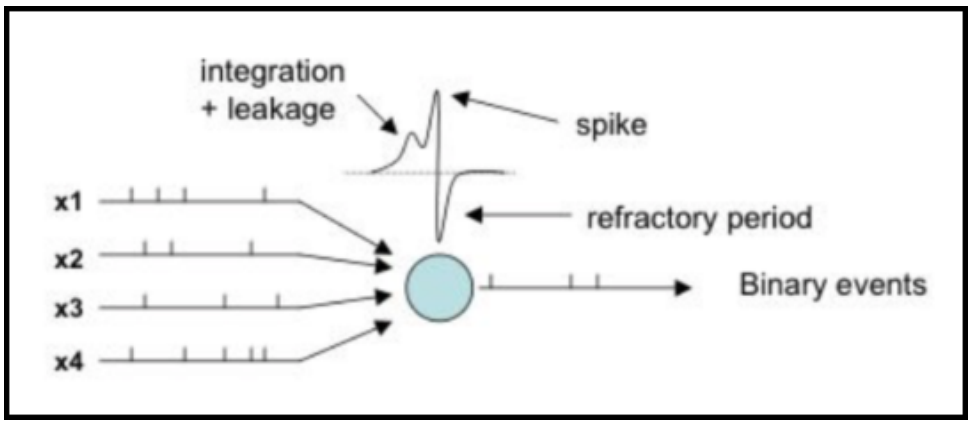

Figure 1.1: Basic model of a spiking neuron (Image credit: EPFL)

1.1.1 Advantages of SNN

- Spiking Neural Network's overcome the computational power of neural networks made of threshold. They open up new perspectives for developing models with an exponential capacity of memorizing and a strong ability to fast adaptation.Moreover, SNNs add a new dimension, the temporal axis, to the representation capacity and the processing abilities of neural networks[3].

- The principal difference between a traditional ANN and SNN is the neuron model that is used. In a traditional ANN, it does not engage individual spikes in computations. Instead the output signals generated from the neurons are treated as normalised firing rates. This is an averaging procedure and it is called rate coding. Instead of using rate coding, SNN uses pulse coding. Each individual spike is used in the neuron model of an SNN. The main characteristic here is incorporation of timing of the firing in computations, like real neurons do.

- In traditional ANN, one of the most common learning algorithm is the Stochastic Gradient Descent. To calculate the gradient of the loss function with respect to the weights, most state of the art ANNs use a procedure called back-propagation. However, the biological plausibility of back-propagation remains highly debatable. For example, there is no evidence of a global error minimisation mechanism biological neurons[3] .

1.1.2 Pre-Post synapse mechanism

One of the major features of a spiking neuron is the membrane potential, the transmission of an individual spike from one neuron to another is mediated by synapses. A pre-synaptic neuron is a transmitting neuron and a postsynaptic neuron as a receiving neuron. In inactive stage the neurons possess a small negative electrical charge of - 70 mV, known as the resting potential. When a single spike arrives into a postsynaptic neuron, it generates a post synaptic potential which is excitatory when the membrane potential is increasing and inhibitory when decreasing.

The membrane potential at an instant is the sum of all present PSP at the neuron inputs. The membrane potential generates a postsynaptic spike after it overcomes a critical threshold value. It further enters the neuron into a refractory period when the membrane remains in an over polarized state consequently preventing generation of new spikes by the neuron temporarily. Post the refractory period, the neuron potential goes back to its resting value and when membrane potential is above the threshold, it fires a new spike.

This figure 1.2 describes different PSP as a function of time and weight value, (a) red line and the blue line is for excitatory, and the green line is inhibitory . (b) Two neurons (yellow) generate spikes, which are presynaptic for next layer neuron (green).Part (c) shows the membrane potential graph for green neuron. Presynaptic spikes raise the potential; when the potential is above threshold, a postsynaptic spike is generated and the neuron becomes over polarized [2].

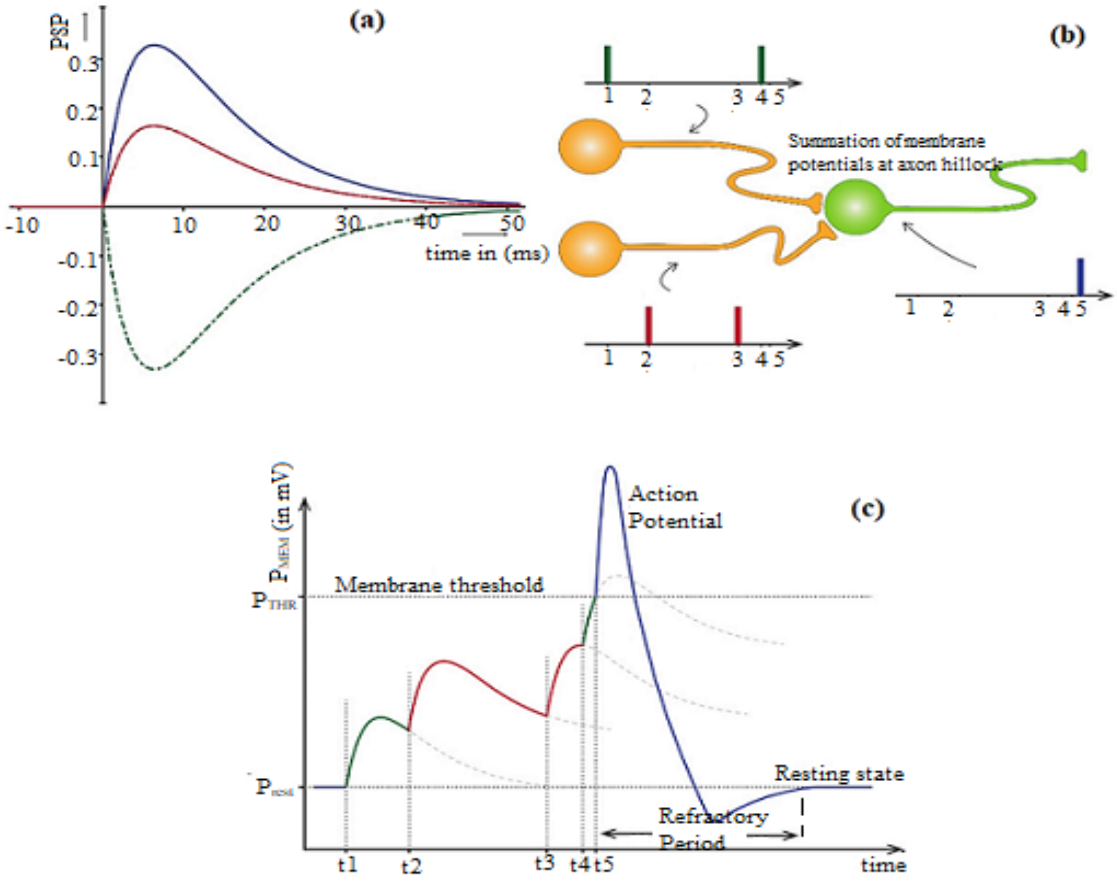

Figure 1.2: Postsynaptic potential function PSP with weight dependency -Source(EURASIP Journal on Image and Video Processing)

1.2 Brain Learning Algorithms

The changing and shaping of neuron connections in our brain is known as synaptic plasticity. Neurons fire, or spike, to signal the presence of the feature that they are tuned for. When two

neurons fire at almost the same time the connections between them are strengthened and thus they become more likely to fire again in the future. When two neurons fire in an uncoordinated manner the connections between them weaken and they are more likely to act independently in the future. This is known as Hebbian learning. The strengthening of synapses is known as Long Term Potentiation (LTP) and the weakening of synaptic strength is known as Long Term Depression (LTD). What determines whether a synapse will undergo LTP or LTD is the timing between the pre- and postsynaptic firing. If the presynaptic neuron fires before the postsynaptic neuron within the preceding 20ms, LTP occurs; and if the presynaptic neuron fires after the postsynaptic neuron within the following 20ms, LTD occurs. This is known as Spike-Timing Dependent Plasticity (STDP) [3].

STDP is an unsupervised learning method. This is also a reason why STDP-based learning is believed to more accurately reflect human learning, given that much of the most important learning we do is experiential and unsupervised, i.e. there is no "right answer" available for the brain to learn from.

There are also a lot of variants of STDP which are undergoing extensive research.It is sometimes cumbersome to maintain the functionality of neural circuits unless, the changes in synaptic strength are coordinated across multiple synapses along with other neuronal properties. Homeostatic plasticity is one of these that includes schemes that control the total synaptic strength of a neuron, that regulate its intrinsic excitability as a function of average activity, or that make the ability of synapses to undergo Hebbian leaning depend upon their history of use. Another variant is STDP + SVM.Unsupervised learning with STDP has been demonstrated with both rate based poisson coding and latency based one spike per neuron coding. Convolution and pooling layers in cascade with the network performing layer by layer learning to learn hierarchical features has shown the best performance reported to date for an unsupervised learning SNN[4]. The Back propagation using STDP model learning rules are inspired from the back-propagation update rules reported for neural networks that are equipped with ReLU activation function[5].

The following is the learning curve for STDP, here the curve has strong depression value than potentiation, increasing specificity[2].

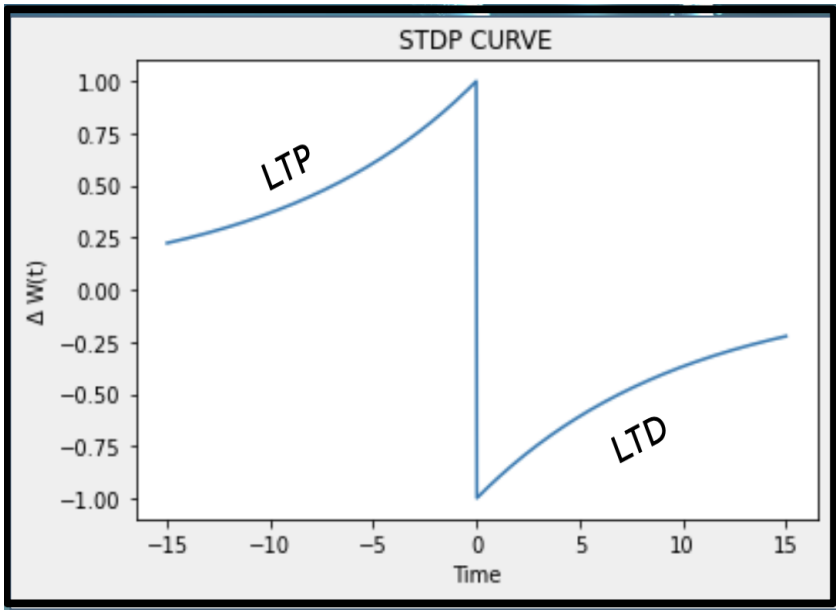

Figure 1.3: The STDP learning curve

The following are the equations used in the weight updation rule for STDP learning algorithm [2]:

$$STDP(\Delta(t)) = \Delta(w) = \begin{cases} A^{-}e^{-\Delta t/\tau^{+}}, & if \Delta t \geq 0 \\ A^{+}e^{\Delta t/\tau^{-}}, & if \Delta t < 0 \end{cases}$$

$$w_{new} = \begin{cases} w_{old} + \sigma \Delta w(w_{max} - w_{old}), & if \Delta w > 0 \\ w_{old} + \sigma \Delta w(w_{old} - w_{min}), & if \Delta w \leq 0 \end{cases}$$

1.3 Motivation

Spiking Neural Networks (SNNs) show a lot of potential as a candidate for human brain motivated neuromorphic evaluation because of their innate power efficiency and noteworthy deduction accuracy for many cognitive problems like image classification and speech recognition. The human brain has robust unsupervised learning ability that is it can function without a data samples that have already been labelled. Therefore, to corroborate brain-inspired neural networks have gained profound interest. The signals is transmitted as spike sequences whereas the conventional neural networks consists of numerical data. Thus, it is not astounding that the at present most famous models in machine learning, artificial neural networks (ANN) or deep neural networks are enlivened by highlights found in biology.The thought of STDP has been demonstrated to be a verified learning algorithm for forward-connected artificial neural networks in numerous applications. To name a few-pattern recognition, Recognising traffic, sound or movement using Dynamic Vision Sensor (DVS) cameras, gender recognition are the related areas that have been already explored.

1.4 Problem Statement

- To implement the Spiking Neural Network architecture for image classification.
- To perform STDP as our brain learning algorithm, which will enable us to update the synapses efficiently.
- To be able to execute the reconstruction of the input image.

Chapter 2

Literature Survey

M. M. Taylor in the year 1973 had suggested that when Hebbian learning occurred that is related to strengthening of synapses for which a presynaptic spike occurred just before a postsynaptic spike, while in anti-Hebbian learning synapses weakened, without the absence of a closely timed presynaptic. Currently, one such research project is the Human Brain Project which brings together neuroscientists, computer and robotics experts to build a unique Information and Communications Technology (ICT)-based infrastructure for brain research. The Human Brain Project is in association with ongoing initiatives in Europe and beyond USA, Canada and Japan. They aim to boost the understanding of brain through modelling and simulation in computers.

Peter U. Diehl et.al [6], presented a SNN which relies on a combination of biologically plausible mechanisms and uses unsupervised learning.Dataset considered for training and testing for the proposed SNN is MNIST dataset. The performance of the approach reported as 95 $\%$ using 6400 learning neurons. They have observed that later inhibition has generated competion among neurons and homoeostasis helps in giving each neuron an equal chance to compete. The input to the network (MNIST) dataset, it contains 60,000 training examples and 10,000 test examples with the input images having pixel size of 28x28 of digits 0-9. The input is in form of Poisson spike trains having the firing rates proportional to intesity of pixel images. Those Poisson-spike trains are fed as input to excitatory neurons. These neurons are connected to inhibitory neurons via one-to-one connections. This connectivity provides lateral inhibition and leads to competition among excitatory neurons. The presented network on the MNIST dataset achieved good classification performance using SNNs with unsupervised learning made of biologically plausible components.

The authors Peter Diehl and Matthew Cook [5], propose a temporally local learning rule that uses the backpropagation weight change updates applied at every time step. The aforementioned techique has two folds benefits an accurate gradient- descent and temporally local, efficient STDP. The above approach is experimented on three different data sets, namely the XOR problem, the Iris data, and the MNIST dataset. The following results were obtained in the paper -STDP-based backpropagation 3-layer SNN; H1=500; H2=150, where H1 and

H2 are the number of neurons in the hidden layers with an accuracy of 97.20%. Therefore, a network of spiking IF neurons can undergo backpropagation learning applied to conventional NNs demonstrating along with the accuracy obtained with the datasets show that SNN performs as successfully as the traditional NNs. SNN receives spike trains representing input feature values in T ms. The learning rules and the network status in the SNN are specified by an additional term as time (t)The major difference between the shown two networks is their data communication where the neural network (left) receives and generates real numbers while the SNN (right) receives and generates spike trains in T ms time intervals.[5].

Reference [2], a novel, simplified and computationally efficient model of spike response model (SRM) neuron with spike-time dependent plasticity (STDP) learning is presented. The data representation is done by frequency spike coding based on receptive fields analogous to visual cortex the images are processed and encoded by the network. The network output proves its utility as primary feature extractor for further refined recognition or as a simple object classifier. The results display that the model can successfully learn and classify black and white images with added noise or partially obscured samples with up to 20 computing speed-up at an equivalent classification ratio when compared to classic SRM neuron membrane models. The proposed solution in the paper is an amalgamation of spike encoding, network topology, neuron membrane model and STDP learning.

Chapter 3

Methodology

3.1 Overview of Network Structure

This chapter describes the building of network elements used in making the Spiking Neural Network. There are 4 network elements. The first is neuron, which is generated by feeding it a random spike train and the membrane potential. The second is synapse, it is the weighted path, which at first is randomly initialised and then updated using STDP .A simple network layer is implemented that consists of 2 layers- the first contains 784 neurons and the second layer(hidden) is made up of 3 neurons. Synapse connects all the neurons in the first layer to every neuron in the second layer. The third is receptive field, in which we make a sliding window 5x5. This receptive field was weighted according to Manhattan distance.The fourth element generated is the spike train, whose frequency is proportional to the membrane potential. Overview of the network structure, 3.1 is depicted in the figure below:

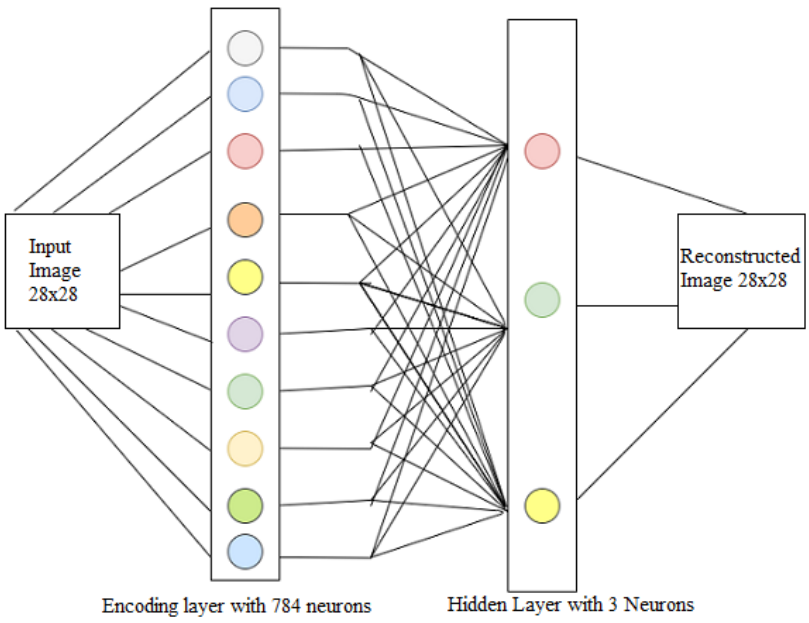

Figure 3.1: Overview of the Network Structure

3.2 Network Elements

The python implementation of hardware efficient spiking neural network .The aim is to build a network which could be used for prediction and on chip-learning.The network elements are shown in the figure 3.2.

- Image Encoding
- Neurons
- Synapse

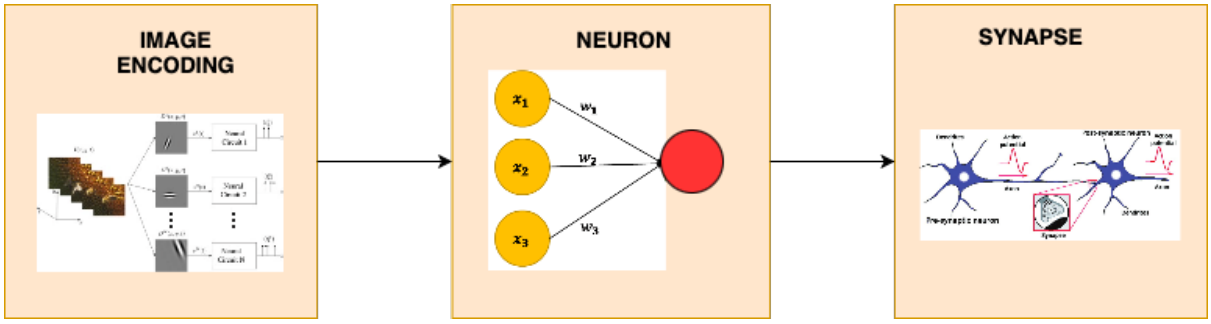

Figure 3.2: Network Elements Flow Diagram

3.2.1 Image Encoding

The figure 3.3 illustrates the conversion of an image to spike train. The encoding an image involves the conversion of the pixel values to spike train [7].

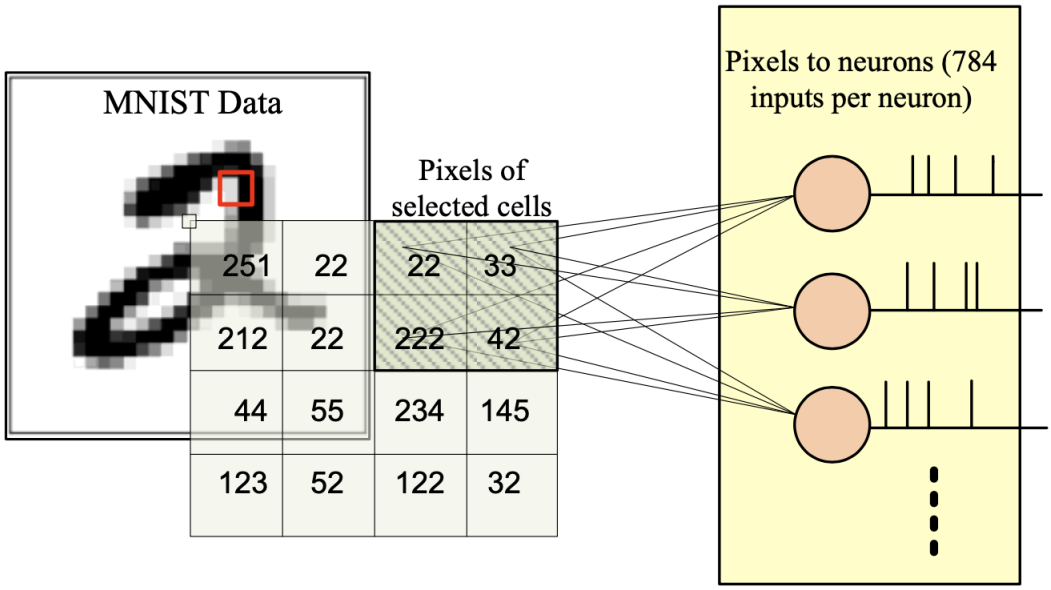

Figure 3.3: Neuromorphic spike encoding mechanism of input image

3.2.2 Synapse and Learning Algorithm

In neurobiology synapse is a junction between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter. In an SNN, synapse is the weighted path for generated spikes from one neuron to the other connected neurons. This is the implementation of a simple network of 784 layers with neurons in the first layer and 3 in the second. Each neuron in the first layer is connected to all the neurons in the second layer via synapse. Synapses are realised by a 2D matrix of size (3x784) initialised with random weights. It can be expanded to any number of layers with any number of neurons in it. Here, we have implemented a two layer network.The input of the hidden layer is the dot product of the output of the encoding layer, the spike train and the synapse which was initialised, randomly at first. These synapses are later updated using our brain learning algorithm, STDP.

3.2.3 Neuron Model

Neuron is the fundamental unit of an SNN. The input, hidden and output layers consist of several interconnected neurons and this neuron emulates the classic leaky integrate-and-fire (LIF) model [8]. Defining membrane potential P_t as a function of incoming spikes and times. Since the model is developed to be utilised in digital circuits, the time units are taken in discrete time form. For a n-input SNN taking the time period as non-refractory,every individual incoming input spike S_{it}, i=[1..n] burgeons the membrane potential Pt with respect to the value of the synapse weight W_i. Additionally, the membrane potential decreases by a constant value D, for every time instant. The process is defined by the equation given below−

$$\mathrm{P}_t = \begin{cases} P_{t-1} - \mathrm{D} + \sum_{n=1}^{n} W_i \, \mathrm{S}_{it} \, , & if P_{min} < \mathrm{P}_{t-1} < \mathrm{P}_{threshold} \\ P_{refract}, & if P_{min} \geq P_{threshold} \\ R_p \, , & if P_{t-1} \leq P_{min} \end{cases}$$

When the potential crosses a threshold value, neuron enters into refractory period in which no new input is allowed and the potential remains constant. To avoid strong negative polarization of membrane, its potential is limited by Pmin. As long as $P(n) > P(\min)$, there is a constant leakage of potential.

The neuron implementation was done in python using the parameters specified in table 3.1, a random spike train was given as input to neuron and after defining the parameters of the above equation, the same loop was implemented.

Parameters	Values	Description
$pixelx$	28	Pixel value of input image
m	28x28	No. of neurons in first layer
n	3	No.of neurons in hidden layer
P_{ref}	0	Refractory Potential
P_{rest}	0	Rest Potential
P_{min}	-500	Minimum Potential
P_{th}	25	Threshold Potential
D	0.25	Leakage Factor
t_{ref}	30	Refractory Time
t_{rest}	-1	Rest Time

Table 3.1: Various Parameters used in the code along with their values

3.3 Processing Workflow

The figure 3.4 delineates the work flow of the implemented code which together constitute the spiking neural network.

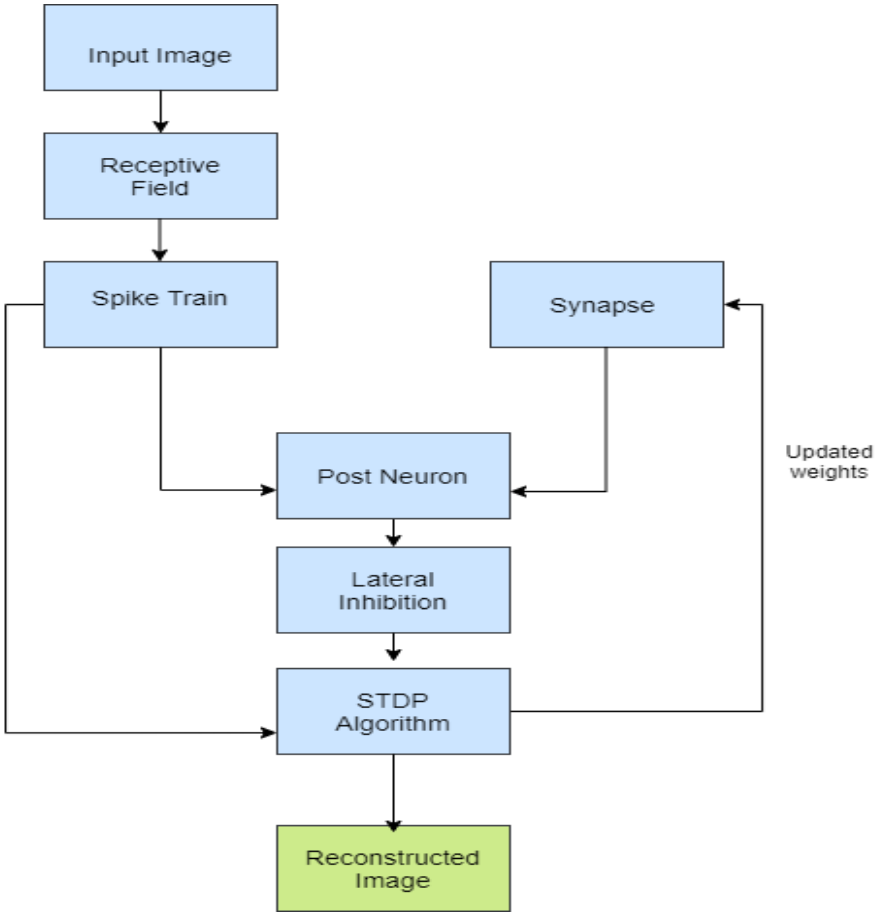

Figure 3.4: The workflow and the main components of the spiking neural network

3.3.1 Receptive Field

Stimulation leads to response of a particular sensory neuron in an area which is known as the receptive field. For spiking neural network, where the image is the input, the receptive field of a sensory neuron is the part of the image which increases its membrane potential. In our implementation, we have used on-centred field. The on-centered receptive field which we have used is 5x5. This receptive field was weighted according to the Manhattan Distance to the center of the field. A 28x28 pixel input is processed by a 28x28 encoding neuron layer(784 neurons), obtaining a potential value for each input which will further be converted into spikes[2].In python, after calculating the Manhattan distance, the membrane potential for each of the 784 neurons was calculated and the output obtained was a 28x28 matrix.The table 3.2 shows the various parameters used in simulation and the figure 3.5 outlines the code structure.

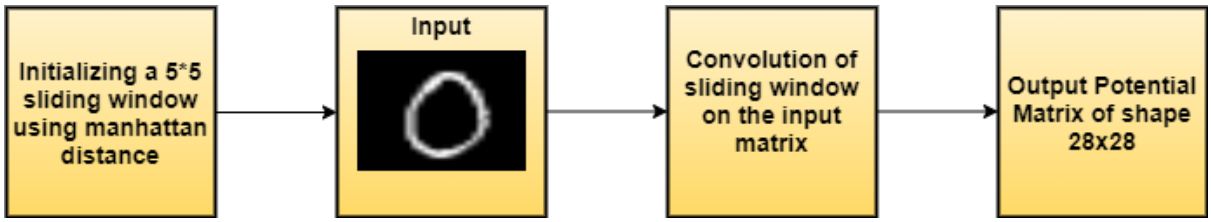

Figure 3.5: Workflow Diagram of Receptive Field

Parameters	Values	Description
ox	2	origin distance from x-axis
oy	2	origin distance from y-axis

Table 3.2: Various Parameters used in the code along with their values

3.3.2 Spike Train

Data processing in Spiking Neural Networks takes place in the form of a series which are called the spike trains. The spike trains comprise of a binary computation and the values oscillating between in terms of between binary logical levels of '0' and '1'. SNN are able to process temporal patterns, not only spatial, and SNN are more computationally powerful than ANN The stimulus by receptive field is fed into the input layer of the neuron. The Stimulus value computed using the sliding window is an analog value and has to be converted into a spike train so that neuron can understand it. This encoder serves as an interface between numerical data (from the physical world, digital simulations, etc) and SNNs.The output of the receptive field,the membrane potential matrix, is converted into spike trains, whose frequency is proportional to the potential. To calculate this frequency, the interpolation function in python was utilised.The table 3.3 shows the various parameters used in simulation and the figure 3.6 shows

the process

Parameters	Values	Description
f_{min}	-1.069	Minimum potential Value
f_{max}	2.781	Maximum potential Value

Table 3.3: Various Parameters used in the code along with their values

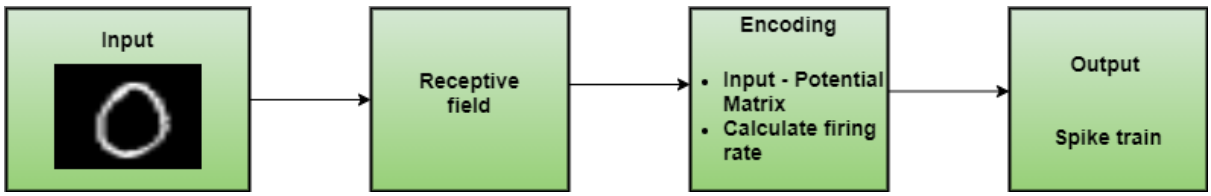

Figure 3.6: Workflow Diagram of Spike Train

3.3.3 Lateral Inhibition

Lateral inhibition is the occurrence in which a neuron's response to a stimulus is inhibited by the excitation of a neighboring neuron. Lateral inhibition impairs the dispersion of action potentials from excited neurons to neighboring neurons in the lateral direction. This property is also called as Winner-Takes-All (WTA)[2]. Unsupervised learning requires competition, hence, lateral inhibition was introduced and thus, the weights of the first spiking neurons, table 3.4 , or the winner neurons, are increased and the other non spiking neurons experience a reduction in their weight value. This is an optimising technique, because tests have showed that depressing the weights of the non spiking neurons makes the network more robust by decreasing the amount of noise in the system.

Parameters	Values	Description
f_{spike}	0	First Spike

Table 3.4: Various Parameters used in the code along with their values

3.3.4 Reconstruction

In a spike based network environment, inputs can be transformed into temporal spike maps, which are then utilised to reconstruct the input image. The input images of the MNIST dataset are trained by the Spiking Neural Network to reconstruct our input image back. The figure 3.7 outlines the logic of code and table 3.5 states the parameter values.

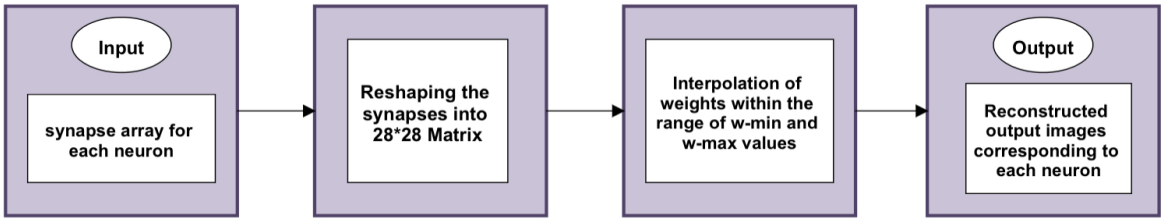

Figure 3.7: Workflow Diagram of the Reconstruction of Synapse

Parameters	Values	Description
w_{min}	-1.5	Maximum value of synapse weights
w_{max}	1.2	Minimum value of synapse weights

Table 3.5: Various Parameters used in the code along with their values

3.4 Image Data-set

The MNIST database of handwritten digits contains 60,000 and 10,000 training and test samples respectively. The digits have been size-normalized and centered in a fixed-size image.The data set is useful for learning techniques and pattern recognition testing on real-world data an it doesn't a lot of pre-processing and formatting. Each image is of the size 28 x 28 Pixel. The figure 3.8 show some of the hand written digit samples available in the data-set.

(a) MNIST dataset image-1 (b) MNIST dataset image-2

Figure 3.8: MNIST dataset Image samples

Chapter 4

Results and discussions

This chapter entails all the graphs and outputs we observed after implementing our code.

4.1 Receptive Field Output

The figure 4.1 shows the output of the receptive field function -a matrix for 784 neurons of the dimension 28 * 28 which contains all the potential values.

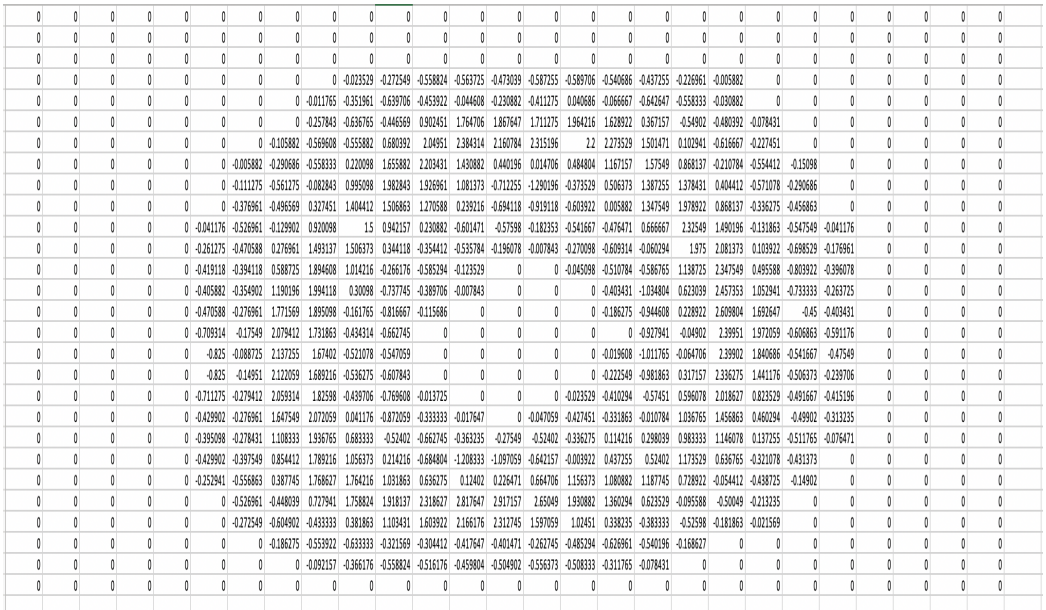

Figure 4.1: Receptive Field Output - Membrane Potential Matrix (28 X 28)

4.2 Spike Train Output

The figure 4.2 depicts the spike pattern across a neuron group over a period of time. To elucidate, if the third neuron spikes at a ninth time unit then there will be a point in row 3, column 9.

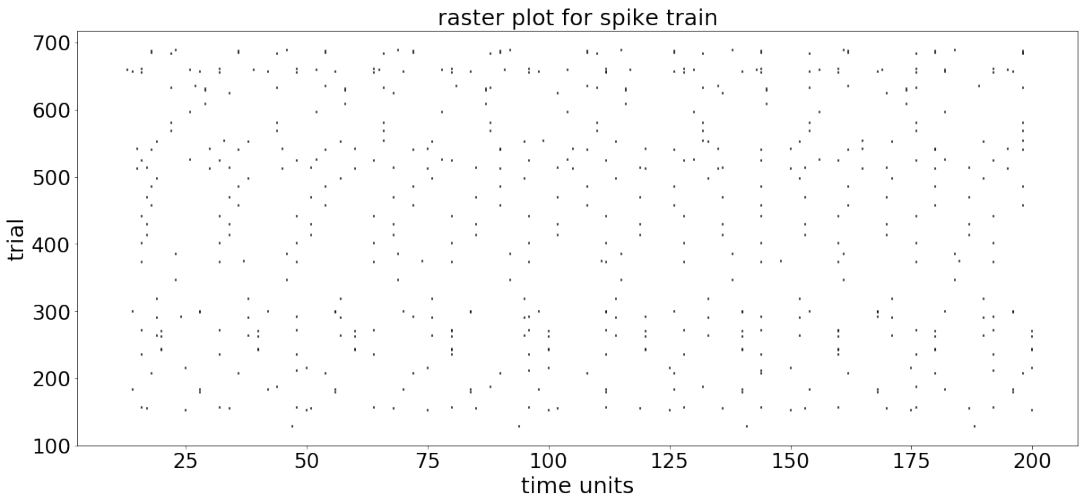

Figure 4.2: Raster Plot of the spike train

4.3 Potential Array Output

The figure 4.3 shows the output of the receptive field function -a matrix for 784 neurons of the dimension 28 * 28 which contains the potential values of all the neurons run for 12 epochs.The value is slowly integrating till it reaches about 14mV and we get our first post-synaptic spike at 25TU, after which it enters the refractory period of 30TU as mentioned above. The neuron does not fire during the refractory period and hence, no spike is observed then.

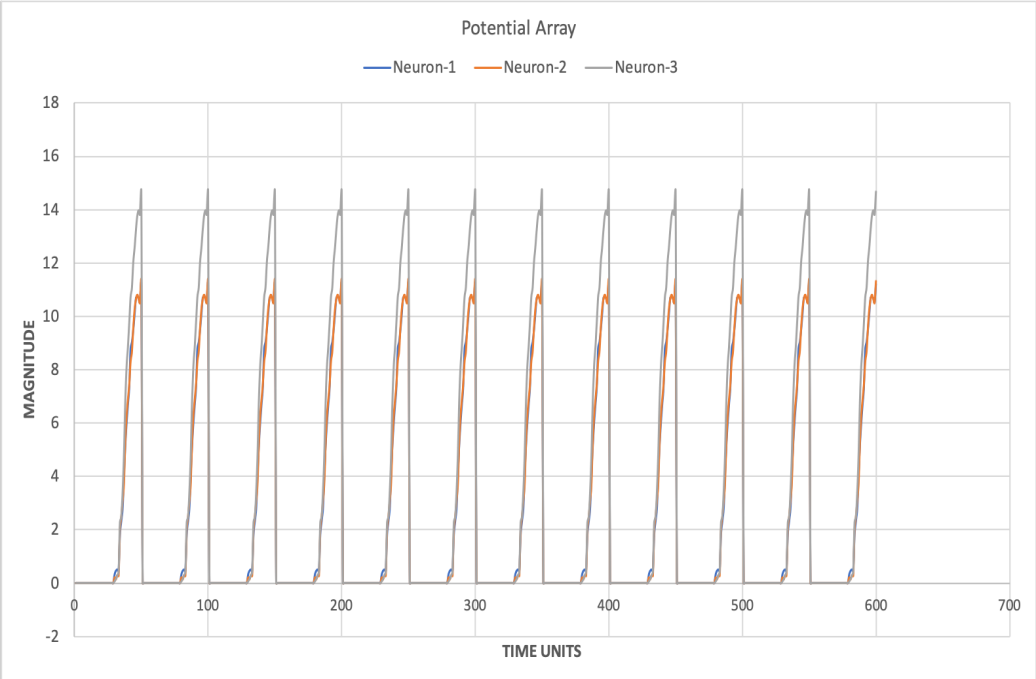

Figure 4.3: Potential Array Graph

4.4 Updated Synapse Output

The figure 4.4 shows synapse update of the neuron-2 which is the winner neuron in that case and a difference between the initial and final synapse can be observed in the graph, which indicates a synapse update corresponding to the reinforcement learning.

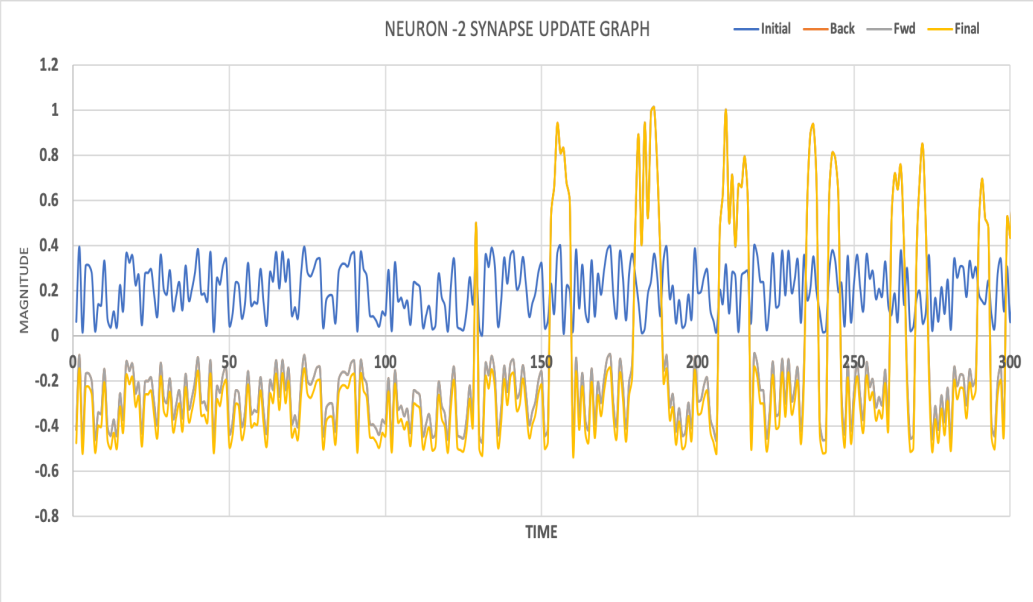

Figure 4.4: Updated synapse Graph

4.5 Reconstructed Images Output

Figures 4.5 and 4.6 depict our final reconstructed images. Reconstruction was carried out using the SNN architecture and employing brain learning algorithm, STDP. For an input image, we have received three neuron outputs as the hidden layer had three neurons. According to the results shown below, our winner neuron was the second neuron for 4.5 , and the rest two neurons gave noise as the output, because their weights were depressed by lateral inhibition technique which optimises our network. In figure 4.6 the winner neuron is the first neuron and a reconstructed image similar to the input image is obtained.

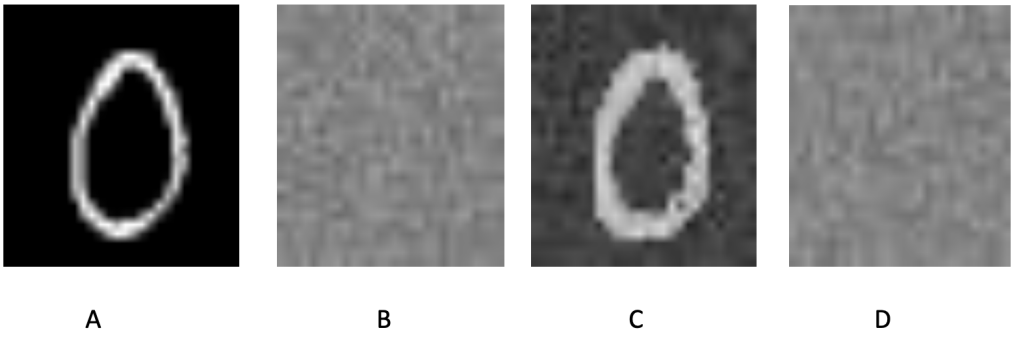

Figure 4.5: Reconstructed images B-D for input image A

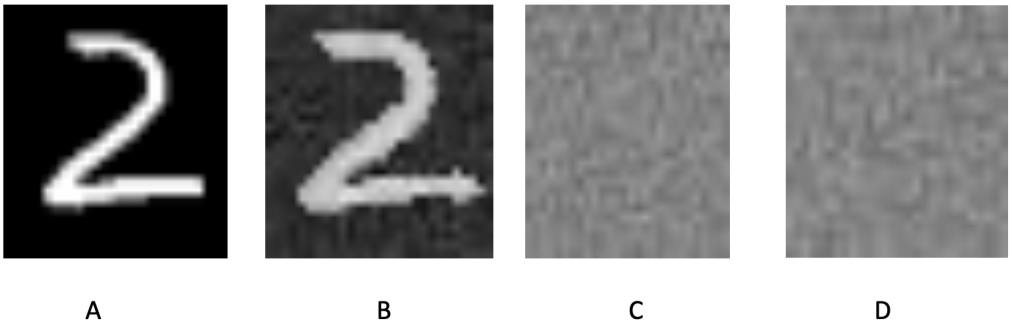

Figure 4.6: Reconstructed images B-D for input image A

Chapter 5

Conclusion and Future scope

In this project, we implemented a simplified spiking neural network (SNN) for image classification using python. The implementation of SNN majorly depends on three network modules and are described as follows: (1) image encoding, (2) synapse and learning algorithm (3) Neuron model. Each module is created individually as a class in python. Image encoding is done using receptive field (RF). This eventually generates spike train for the given input image. MNIST dataset is considered as input images for image classification. Initially random weights were created and used to process the post neuron and subsequently STDP learning algorithm is used to update the winner neuron. This work adopts the winner-take-all model to ensure only one neuron will be highlighted with highest post-synaptic potential value. This can be done with lateral inhibition process i.e. remaining neuron membrane potential will be suppressed. Finally, we reconstruct the image with the updated synapse weight. Network preserves the learning and the classification properties, computational time and complexity is reduced. Therefore, it can be corroborated that STDP learning is efficient, robust, and can work well with noisy data.

SNN implementation needs further optimization with advance brain learning algorithms available in the literature. This will help in reducing complexity for image classification. Later, the optimized SNN will provide RGB images as input which can be used for attendance management application where facial recognition of students in online learning can be done using the model and their attendance can be marked.

Bibliography

[1] R. Gopalakrishnan, Y. Chua, and L. R. Iyer. Classifying neuromorphic data using a deep learning framework for image classification. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1520–1524, 2018.

[2] Juan F Guerrero-Martínez Manuel Bataller-Mompeán Taras Iakymchuk, Alfredo Rosado-Muñoz and Jose V Francés-Víllora. Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing*, 2015.

[3] Yi-Ling Hwong. Unsupervised learning with spike-timing dependent plasticity.

[4] A. Vasudevan, T. Serrano-Gotarredona, and B. Linares-Barranco. Learning weights with stdp to build prototype images for classification. In *2019 14th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, pages 1–5, 2019.

[5] Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39 – 47, 2019.

[6] Peter Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015.

[7] Sonal Singhal Siona Menezes Picardo4 Nilesh Goel Aadhitiya VS, Jani Babu Shaik. Design and mathematical modelling of inter spike interval of temporal neuromorphic encoder for image recognition. 2020.

[8] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

Appendices

Appendix A

Code Attachments

A.1 Main.Py

Main.py is the main file of the code where all the other python scripts are called and executed in a sequential manner. The four components Neurons,synapse, Receptive fields scripts are called and their outputs serve as the input of the next component. At the end of the main.py script the updated synapses are obtained and reconstruction of the synapse weights take place

```
import numpy as np
from neuron import neuron
import random
from rl import *
# from matplotlib import pyplot as plt
from recep_field import rf
import cv2
from spike_train import encode
from parameters import param as par
from var_th import threshold
import os
import time

#potentials of output neurons
pot_arrays = []
#appending 3 new arrays
for i in range(par.n):
  pot_arrays.append([])
print(pot_arrays)

#time series
time  = np.arange(1, par.T+1, 1)
print("time, ", time)

layer2 = [] # hidden layer ( containg 3 neurons )

# creating the hidden layer of neurons
for i in range(par.n):
  a = neuron()
  layer2.append(a)

#synapse matrix initialization
synapse = np.zeros((par.n,par.m))
# print("SYNAPSE INTIALIZATION",synapse )
```

```
# print("shape of synapse ", synapse.shape)

for i in range(par.n): # 3 times
  for j in range(par.m): # 784 times
    synapse[i][j] = random.uniform(0,0.4*par.scale)



np.savetxt("Init_synapse.csv",synapse,delimiter=",",fmt='%s')
temp_list=[]
temp_list1=[]
for k in range(par.epoch):
  for i in range(322,323):
    print (i,"  ",k)
    img = cv2.imread("/Users/pranika/Desktop/SNN_minor_project/
    Dataset_Images/iris.png", 0)

    #Convolving image with receptive field
    pot = rf(img)
    # print(f"potential matrix , {pot}")

    #Generating spike train
    train = np.array(encode(pot))

    #calculating threshold value for the image
    var_threshold = threshold(train)
    print(var_threshold)
    var_D = 0.15*par.scale
    print(f"Variable threshold {var_threshold}")
    for x in layer2:

      x.initial(var_threshold)
      print(x)

    #flag for lateral inhibition
    f_spike = 0 # First spike
    img_win = 100

    active_pot = []
    for index1 in range(par.n):
      active_pot.append(0)
      print(active_pot)

    print("SYNAPSE")
    print(synapse)
    print(synapse.shape)

    c=0

  #Leaky integrate and fire neuron dynamics
    for t in time:
      for j, x in enumerate(layer2):
        active = []

        if(x.t_rest<t):
          x.P = x.P + np.dot(synapse[j], train[:,t]) # dot product of two
    arrays

```

```
            print("train shape",train.shape)
            print("-------------------------")
            print(f"x.P for {j} is {x.P}")
            print("par.Prest", par.Prest)
            if(x.P>par.Prest):
              print(f"x.p before decrease -  {x.P}")
              x.P -= var_D
              # c=c+1
              print(f"x.p after decrease -  {x.P}")
              print("-------------------------")
            # print("counter value", c)
            active_pot[j] = x.P

            # temp_list.append(x.P)

          pot_arrays[j].append(x.P)
          np.savetxt("pot_arrays.csv",pot_arrays,delimiter=",",fmt='%s')
# print(len(pot_arrays))
# print("POT_ARRAY",pot_arrays)
# print("active pot",active_pot)
# Lateral Inhibition
        print("working")
        if(f_spike==0):
          high_pot = max(active_pot)
          print("var_thresh",var_threshold)
          if(high_pot>var_threshold):
            f_spike = 1
            winner = np.argmax(active_pot)

# print("Winner Value",winner)
            img_win = winner
            print ("winner is " + str(winner))

            for s in range(par.n):
              if(s!=winner):
                layer2[s].P = par.Pmin

                temp_list1.append(layer2[0].P)
                temp_list2.append(layer2[1].P)
                temp_list3.append(layer2[2].P)
        print("layer2_len",len(layer2))
        temp_list.append(f_spike)
# print("value=",layer2[0].P)

        #Check for spikes and update weights
        for j,x in enumerate(layer2):
          print(1)
          s = x.check()
          if(s==1):
            x.t_rest = t + x.t_ref
            x.P = par.Prest
            for h in range(par.m):
              for t1 in range(-2,par.t_back-1, -1):
                if 0<=t+t1<par.T+1:
                  if train[h][t+t1] == 1:
                    # print "weight change by" + str(update(synapse[j][h], rl
    (t1)))
                    synapse[j][h] = update(synapse[j][h], rl(t1))
```

```
                else:
                  synapse[j][h]=synapse[j][h]
                np.savetxt("Back_up_synapse.csv",synapse,delimiter=",",fmt='%s')
                print("22")


            for t1 in range(2,par.t_fore+1, 1):
              print("TIME",t)
              if 0<=t+t1<par.T+1:
                if train[h][t+t1] == 1:
                  # print "weight change by" + str(update(synapse[j][h], rl(t1)))
                  synapse[j][h] = update(synapse[j][h], rl(t1))
                else:
                  synapse[j][h]=synapse[j][h]
                np.savetxt("FWD_up_synapse.csv",synapse,delimiter=",",fmt='%s')
                print("11")

     if(img_win!=100):
       for p in range(par.m):
         if sum(train[p])==0:
           synapse[img_win][p] -= 0.06*par.scale
           if(synapse[img_win][p]<par.w_min):
             synapse[img_win][p] = par.w_min
```

A.2 Neuron.py

```
import numpy as np
import random
from matplotlib import pyplot as plt
from parameters import param as par

class neuron:

  def __init__(self):
    self.t_ref = 30
    self.t_rest = -1
    self.P = par.Prest

  def check(self):
    if self.P>= self.Pth:
      self.P = par.Prest
      return 1
    elif self.P < par.Pmin:
      self.P  = par.Prest
      return 0
    else:
      return 0

  def inhibit(self):
    self.P  = par.Pmin

  def initial(self, th):
    self.Pth = th
```

```
28      self.t_rest = -1
29      self.P = par.Prest # -> 0
```

A.3 ReceptiveLearning.py

```
1  #STDP reinforcement learning curve
2  def rl(t):
3
4    if t>0:
5      return -par.A_plus*np.exp(-float(t)/par.tau_plus)
6    if t<=0:
7      return  par.A_minus*np.exp(float(t)/par.tau_minus)
8
9
10 #STDP weight update rule
11 def update(w, del_w):
12   if del_w<0:
13     return w + par.sigma*del_w*(w-abs(par.w_min))*par.scale
14   elif del_w>0:
15     return w + par.sigma*del_w*(par.w_max-w)*par.scale
```