Implementing a Spiking Neural Network with Floating-Point neurons into GeNN to compare the performance to that of a Few-Spikes neural network in computer vision.

Student: Thomas Shoesmith [CandNo. 198687]
Supervisor: Thomas Nowotny

Interim Report
Final Year Project
Computer Science with Artificial Intelligence

Department of Informatics
University of Sussex
08/08/2022

1 Statement of Originality

This report is submitted as part requirement for the degree of Computer Science with Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for a copy of this report to be loaned out to students in future years.

Signature: *Thomas Charles Shoesmith*
Date: 08/08/2022

2 Acknowledgements

I wish to acknowledge the support, advice and opportunity that my supervisor, Thomas Nowotny, has provided to me over the duration of this Final Year Project. The help and assistance given by himself and the GeNN Team have greatly broadened my understanding of the GeNN framework, with an enthusiasm now focusing on pursuing this line of research.

I also wish to acknowledge the friends who have made the university experience as wonderful and pleasant as it has been. Especially to James Standen, a close friend who I met on the course, to which we spent countless hours both inside and outside of university, but unfortunately passed away early this year. Never lose a chance of saying a kind word.

3 Summary

Despite yielding impressive results from image classification to playing against humans in games such as AlphaGo[1], Artificial Neural Networks (ANN) require a significant amount of energy when compared to the human brain. AlphaGo, as an example, required approximately 1MW of power to run[2], meanwhile, the energy consumed by an average human brain is equivalent to approximately 20W[3] resulting in Alpha go having 50,000 times higher energy consumption than the average human brain in this competition. This, therefore, brings us onto the latest challenge in modern machine learning, the focus on the efficiency of algorithms, so that machine learning can be run on more portable devices or scaled up without the requirement of large energy resources.

Spiking Neural Networks (SNN) have demonstrated the potential to offer a more energy efficient alternative, compared to ANN, mainly due to their lower computational requirements and the sparser spike-based communications between neurons[4]. However, training large SNN from scratch remains challenging and converting a high performing ANN into a similarly performing SNN has typically involved encoding ANN activations in the firing rate of the spiking neurons[5]. The resulting SNN from this conversion typically requires a relatively greater number of spikes to generate a similar performance, with the larger numbers of spikes outweighing the energy savings of the SNN.

In a recent paper by Christoph Stöckl and Wolfgang Maass[6], they introduce a new method of ANN to SNN conversion titled: "Few Spikes (FS) conversion". FS treats each neuron's output spikes as a binary code representing the activation of the ANN neuron and, by using the timing of the spikes to hold additional information, fewer spikes are needed and each input can be presented using fewer time steps.

This project will build on the research from this paper to explore further optimisations to these binary codes, potentially allowing the number of spikes and time steps to be further reduced and overall efficiency improved. The SNN performance will be evaluated through the accuracy of image classification on a selection of benchmark data sets including MNIST [7], CIFAR-10[8] and Omniglot[9].

Contents

Abstract

Described as the third generation of machine learning, Spiking Neural Networks have demonstrated the possibility of having high performance machine learning algorithms using a fraction of the power compared to their conventional machine learning counterparts. The majority of the power consumption and efficiency loss of typical neural networks comes from the retrieval of information from registries on Von Neumann architecture machines. Spiking Neural Networks conversely, on optimal neuromorphic architecture, are able to dramatically reduce this problem due to their analog, bio-inspired architecture.

However, with this new generation of machine learning architecture comes a new style of coding the neural networks. Spiking Neural Networks takes on an even greater bio-inspired approach than current approaches with more realistic models of neurons and synapse behaviour.

Developing new neural coding methods therefore would greatly improve this area of research and potentially develop greater and more efficient Spiking Neural Networks.

4 Introduction

This project will be investigating a similar, but alternative temporal coding method based off the Few-Spikes conversion developed by Christoph Stöckl and Wolfgang Maass[6]. The Few-Spikes (FS) conversion uses spikes to simulate binary representation, allowing for a significant reduction in spikes required, this temporal approach versus the typical rate-based encoding techniques has demonstrated a strong and promising performance in the form of value representation accuracy with a significantly reduced rate of spikes.

to represent a rational number in binary, there are two typical approaches: fixed-point representation and floating-point representation. Few-Spikes conversion uses a fixed-point representation equivalent to convert existing decimal values to spikes where the range is traded for precision by not requiring data to hold information about the scale of the data. As discussed before, this approach has demonstrated significant strides in Spiking neural network coding, however, the compromise on range potentially limits the performance of this method generally.

Floating-point (FP) representation as an alternative approach to Few-Spikes conversion is where this project is primarily focused. Building on the inspiration and results from FS, FP could potentially offer an improved alternative due to the theoretical greater range of value representation.

Investigating, developing, and testing this new theoretical FP approach and comparing the results to the FS method would require developing and testing the current FS method, developing and testing a FP model, then comparing the two approaches head-to-head. However, it would be beneficial to also test these two approaches at a theoretical level where the range and precision of the values needing to be represented can be tested, this will be investigated first.

This report therefore will first investigate the theoretical advantages and potential disadvantages of this new approach across different datasets and value representation. The FS model will then first be developed and tested against a basic dataset, with further testing against a more demanding dataset. Attempts to develop the FP model will then be started with the testing of the model done against the same datasets which FS was tested on. Extension tasks for this project include looking into primacy coding, a form of neural coding found in the olfaction system. However, this extension task was only researched without any development.

5 Professional considerations

5.1 BCS Code of Conduct

The relevant sections of the BCS Code of Conduct[10] are listed below, with an explanation as to how this will be dealt with across the duration of the project

5.1.1 Public Interest

a. have due regard for public health, privacy, security and wellbeing of others and the environment

The public health and well being would be unaffected by this project due to the focus being on efficiency and performance with no direct interaction between the public and the project. All data used within the project will be sourced from publicly available repositories.

As the project will be looking into the efficiency of alternative coding methods, there is no direct security risks.

b. have due regard for the legitimate rights of Third Parties
This project will primarily be working with GeNN where it will be purely be used for building models within the simulation environment to compare performance of different models. This would not fall within any legitimate rights issues.

c. conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement
All activities will be done at a high level of professional standard. This project will be looking at software performances, so the only relevance this section has would be during interactions with supervisors or other academics.

d. promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise
There will be limited to no opportunities for this section within the project, however, any correspondence will be dealt with equally.

5.1.2 Professional Competence and Integrity

a. only undertake to do work or provide a service that is within your professional competence.
The project proposal and outline has been discussed with the supervisor and the area of research is expected to be within the expected professional competence.

b. NOT claim any level of competence that you do not possess.
This project is a continuation from a University of Sussex JRA, the understanding of the project has been researched and previously practiced. No level of competence has been falsely claimed.

c. develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field.
Throughout this project, research will be required into the fields relevant to the topic, in this case Spiking Neural Networks and GeNN. Correspondence with the project's supervisor and the GeNN GitHub will maintain an up to date understanding on the standards and procedures within this field.

d. ensure that you have the knowledge and understanding of Legislation and that you comply with such Legislation, in carrying out your professional responsibilities.
GeNN is licenced under GNU General Public License, version 2, which allows for changing of the free software and permission to use the code as long as any published works give reference to the original source code. [11]

MNIST is licenced under GNU General Public License, version 3, which allows for changing of the free software and permission to use the code as long as any published works give reference to the original source code. [12]

CIFAR-10 and Omniglot is under a MIT License, which allows for "without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so". [13] [14]

e. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.
Throughout the project, feedback and advice will be requested from both the supervisor as well as other academics within the field, their feedback and support will always be looked at and used to improve upon.

f. avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.
A high professional standard will be maintained throughout this project both within the project and with correspondence. There will be no sensitive information involved that could be mismanaged.

g. reject and will not make any offer of bribery or unethical inducement.
No bribery or unethical inducement will be offered or accepted.

5.2 Ethical Issues

As this project does not involve the collection of data from human participants, there is no need for an Ethical review.

However, there are a couple of topics of discussion around the ethics of the potential application of this project. Whilst this project does not involve any participants or sensitive information, there is a potential risk of the project's outcomes being abused. This project will be looking into the different methods of Spiking Neural Network coding and the difference in both performance and efficiency. Although this directly has practically no ethical issues, these efficient methods could have the potential to be implemented into a system with malicious intent.

This however, is highly unlikely, primarily because this project when completed would be far from application, but also because similar research within this area is ongoing. The data used within this project is all publicly available and the project will also not be published.

5.3 Relevance

Artificial Neural Networks (ANN) make up a significant portion of the Computer Science with Artificial Intelligence degree, it is the building blocks behind computer vision, machine learning, NLP etc. Spiking Neural Networks are an extension of ANN with an aim to increase the efficiency of neural networks; allowing them to be more accessible on smaller, handheld devices or more efficient on scaled up neural networks.

6 Aims and Objectives

6.1 Implement a network of Few Spikes neurons for image recognition

The first objective is to implement an already published method for ANN-SNN conversion using a Few Spikes neuron, within GeNN. This will require two neurons to be developed, one neuron will take a floating-point value, sum up the weights and bias and convert this into a spike train. The second neuron will take a spike train from another Few Spikes neuron, sum up the weights and bias and return the updated spike train.

6.1.1 Start with a more basic data set (i.e. MNIST)

A developed Few Spikes neuron will then be implemented into a Spiking Neural Network. The second objective after developing this neuron would therefore be to implement the Few Spikes neuron into a relatively simple network to be trained on a relatively basic data set. The data set provisionally chosen is the MNIST data set due to several properties.

The first reason is due to its popularity, MNIST is a widely practised data set with an abundance of neural networks implemented to solve image classification problems, therefore this is the ideal data for developing a successful Spiking Neural Network with a Few Spikes neuron. The second reason is the data set size, MNIST contains 60,000 training and 10,000 testing images allowing for plenty of training runs. MNIST images are also a relatively small in size, measuring in at 28 by 28 pixels, allowing for a relatively small neural network.

6.1.2 Finish with a more complex data set (i.e. CIFAR-10 or Omniglot)

After successfully implementing a Few Spikes neuron into a neural network and training it on a relatively basic neural network such as MNIST, the next step will be to implement a Few Spikes neuron into a more complex Spiking Neural Network and develop this on a more complex data set such as CIFAR-10 or Omniglot.

By implementing this neuron into a more complicated neural network, it will allow for a greater demonstration of the neuron's performance compared to later developed neurons.

6.2 Implement a network of floating-point neurons for image recognition

Once a successful development of the Few Spikes neuron and implementation into various neural networks for different data sets is achieved, the next objective is to develop the floating-point neuron within GeNN.

Few Spikes neurons use fixed-point arithmetic to store the value passed through, floating-point neurons have the potential advantage in the use of floating-point arithmetic, allowing for a greater range of value representation.

6.2.1 Start with a more basic data set (i.e. MNIST)

Once the neuron has been developed, it will first be implemented into a relatively basic Spiking Neural Network where it will be used on image classification with the same basic data set used in section 6.1.1.

6.2.2 Finish with a more complex data set (i.e.CIFAR-10 or Omniglot)

Once a successful implementation of the floating-point neuron into a relatively basic image classification has been achieved, the floating-point neuron will need to be implemented into a more complex Spiking Neural Network for training on the same data sets used in section 6.1.2.

6.3 Assess performance of different spike coding methods

Once the four implementations of the two neurons are completed, tests can be conducted comparing the performance between the two neurons in different environments. These tests will compare the accuracy, time taken for training, spike train length (the number of spikes needed to represent a value) and any other metrics which might present themselves as useful.

6.4 Extension

6.4.1 Investigating primacy code as a potential for more efficient neural coding

Another approach into neural coding may come from a more biological influence in the form of primacy code. Odor detection from the olfactory system is achieved regardless of concentration and attention from the source, this ability suggests a unique variant of neural activity, and a potential for a new approach of neural coding. [15]

7 Requirements analysis

7.1 Background Research

7.1.1 Spiking Neural Networks

To understand a Spiking Neural Network, we must first be clear as to what an Artificial Neural Network is.

An Artificial Neural Network is a type of neural network which is a bio-inspired method of machine learning influenced by the biological neural networks found in animal brains. These digital minds are made up of an interconnected group of nodes which form connections with weights and biases. These nodes are often made up of different layers which often provide different functionality. The way these nodes are interconnected in an important feature of the neural network; each connection between nodes holds a bias, and each node holds a weight. A bias is a value that is added onto the signal to be passed through to the next layer whereas a weight controls the amount of influence the oncoming signal has on the node.

Artificial Neural Networks are the current, primary method of machine learning due to their high accuracy and compatibility with current computer architecture; they are able to solve tasks to an accuracy and performance similar to, and sometimes even greater than that of their human counterparts, A good example of this is AlphaGo's victory over Lee Sedol in 2016[16].

However, one area where Artificial Neural Networks do not have an advantage is in their resources requirements, despite yielding relatively impressive results, these neural networks require a relatively large computational resources to operate. AlphaGo was able to defeat Lee Sedol 4-1 in 2016[16], however, when we compare the estimated energy consumption between the two, we see a different side to this story. It is believed the human brain only uses 20W, a relatively tiny amount when compared to AlphaGo's estimated 1MW of power, giving AlphaGo a power advantage by a factor of 50,000.

Spiking Neural Networks offer a potentially more energy efficient method of machine learning. A Spiking Neural Network takes on an even more biological approach to an artificial mind by imitating spikes between nodes instead of sending floating-point values (figure 1). At the most

basic level is a Leaky Integrate and Fire (LIF) neuron (figure 2). This model is a simulation of a simplified biological neuron where there is a membrane voltage, a threshold voltage and a resting voltage. An input is added to the membrane voltage resulting in a variable voltage being added to the membrane voltage, where a large input produces a large voltage, and a small input produces a small voltage. If the added voltage is insufficient to reach the threshold, then said voltage will decay to the resting membrane voltage. If, however, there is a large enough signal (or enough smaller signals that combine to cause the membrane voltage to become equal to or greater than the threshold voltage) then the membrane voltage is reset to the resting voltage, a signal is outputted and there will be a refractory period where another signal cannot be sent.

Both figures have been influenced from an article by Paredes-Vallés[17].

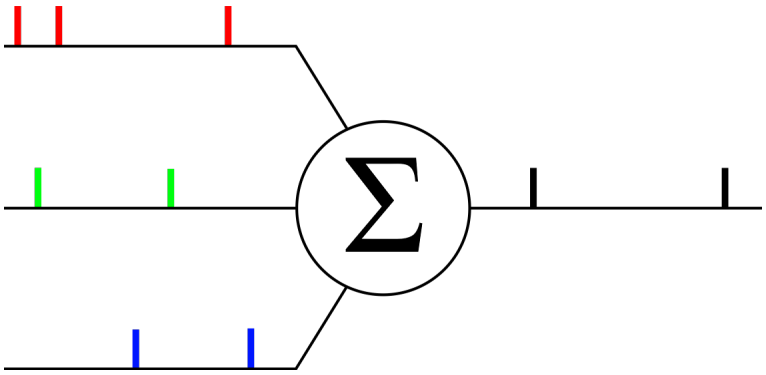

Figure 1: A Spiking Neural Network with inputs from three neurons

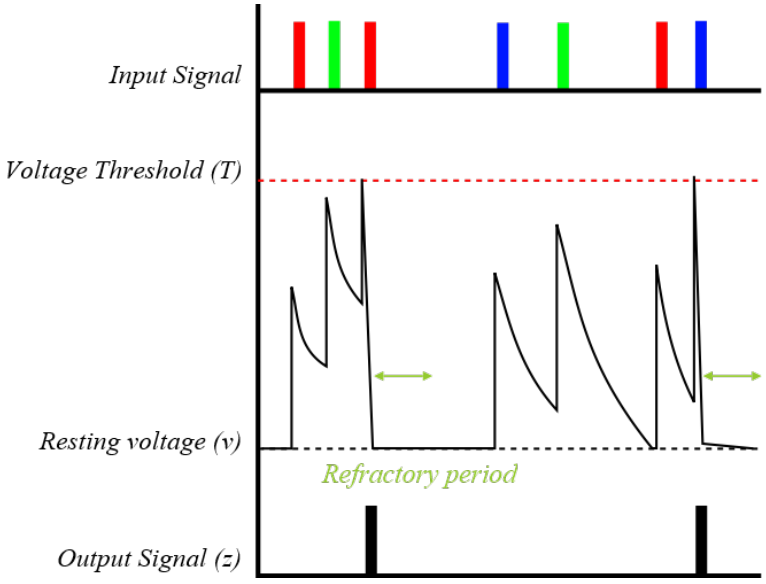

Figure 2: A LIF neuron, taking in inputs from three neurons and outputting a signal once the membrane voltage surpasses the voltage threshold

One area of research within Spiking Neural Networks is the method of coding the spikes between neurons, there are 4 main methods currently being used: Rate Coding, Population coding, Sparse coding and temporal coding.

Temporal coding will be the primary focus of this project. Temporal coding uses the timing of each spike to carry additional information which in turn can reduce the number of spikes needed.

7.1.2 GPU enhanced Neuronal Network

One current disadvantage of Spiking Neural Networks is their inefficiency to run on conventional computer hardware, Spiking Neural Networks reach their optimal performance on neuromorphic

hardware. However, there are simulators available that are capable of running Spiking models on conventional computer architecture.

The University of Sussex have their own Spiking Neural Network simulator titled GPU enhanced Neural Network (GeNN) [18]. GeNN was first proposed in 2016 as a method of using the computational speeds found within graphical processing units (GPUs) to help generate large-scale spiked neural networks with respectable run times.

Since 2016, GPUs can offer general purpose computing thanks to NVIDIA's compute unified device architecture (CUDA), allowing them to be used across a wider range of applications. CUDA allows for C - like code, which can be executed efficiently on NVIDIA's CUDA GPUs [18]. GeNN is able to take advantage of CUDA by compiling some of the code into C, it is able to run significantly faster, allowing for a significant reduction in processing time.

This project will use GeNN to develop neurons that can later be implemented into relatively large Spiking Neural Network models, with the efficiency GeNN is able to produce, model training times hopefully can be of a manageable period.

7.1.3 TensorFlow

TensorFlow[19] is an open source machine learning framework which will be used throughout this project to develop Artificial Neural Networks models which can then be converted via GeNN into Spiking Neural Networks.

TensorFlow is a popular library to use and is supported within GeNN. The most significant note with TensorFlow and GeNN is a couple of compatibility issues. Spiking Neural Networks handle biases differently from ANN, therefore when creating a model, biases will be disabled for each layer. Additionally, Max Pooling, a popular layer choice, but currently not used within GeNN, instead, alternative layers such as average pooling can be used instead.

7.1.4 Few Spikes Conversion Neurons

Directly training a Spiking Neural Network remains a difficult challenge, a current method for producing Spiking Neural Networks is to convert pre-existing Artificial Neural Networks into Spiking Neural Networks by using a variety of conversion method. Few Spikes (FS)[6] offers a conversion method from a floating-point value to a spike train using temporal coding.

A Few Spikes neuron operates in a similar method to the leaky integrate and fire neuron discussed in section 4.1.1 however, the main difference is that with Few Spikes the membrane voltage does not degrade over time and the spikes received / returned are stored in a fixed-point binary representation positioning, rather than frequency.

Within a Few Spikes neuron, there are two global parameters: K and alpha. K is the fixed time frame / spike train length to represent a value. Alpha is the maximum value that can be represented by that neuron / network. Within the neuron, there are several local variables: v, t, T, h and z. v represents the membrane potential, t represents the current time step (this will range from t = 1..K), T represents the threshold value, h is the membrane potential reset and finally, z is the output spike train.

For this project, we will primary be using a ReLU activation function, due to the nature of FS-neurons, the values of T and h will be identical, but for demonstrative purposes, T and h have been defined separately.

The python code for a Few Spikes neuron would be as follows:

```
# Few Spikes Neuron
# x = input
# K = timesteps K = 1, ... t
# alpha = value representation range (cap)

import numpy as np

def fs(x, K = 10, alpha = 25):
    t = 0
    fx = 0
    v = np.zeros(K)
    z = np.zeros(K)

    T = alpha * 2**(-K) * np.array([float(2 ** (K - i))
```

```
                                        for i in range(1, K + 1)]).
    astype(np.float32)
     h = alpha * 2**(-K) * np.array([float(2 ** (K - i))
                                        for i in range(1, K + 1)]).
    astype(np.float32)

     v[0] = x

     while t < K:
         # spike if voltage > threshold, reset if spike.
         if v[t] >= T[t]:
             z[t] = 1
             v[t] = v[t] - h[t]

         # copy over value once reduced.
         if t + 1 < K:
             v[t + 1] =  v[t]

         # sum voltage if spike
         fx += z[t] * h[t] #* 0.5
         t += 1

     return z, fx # outputs tuple of spike train and sum (sum for
    configuration)
```

Listing 1: Python Code for FS-neuron

Few Spikes offers a significant reduction in the number of spikes needed whilst being able to achieve an accuracy similar to that of Artificial neural network models [6].

7.1.5 Floating-Point Conversion Neurons

Floating-point offers an alternative to Few Spikes conversion by using floating-point arithmetic compared to fixed-point arithmetic.

Instead of having the values of T and h being fixed across all neuron within a network, they instead can be changed to the range best covers the value needing to be represented, this has both its advantages and disadvantages. An advantage of using this method is that it allows for smaller floating-point values to have improved accuracy, this is due to the increased range through the use of an exponent. The disadvantages however are that part of the spikes are taken up with the data representing the exponent, this means that the overall accuracy is reduced, especially for larger values, as less spikes are holding information about the value directly (the mantissa).

The methods for coding this neuron style are similar, however the exponent needs to be calculated, and the starting position for the T value also need to be calculated.

The python code for a Floating-point neuron would be as follows (note that in this example, the exponent is sent ahead of the mantissa):

```
# Floating-point Neuron
# x = input
# K = timesteps K = 1, ... t
# elim = length of exponent
# w = weight on neuron
# alpha = value representation range (cap)

import numpy as np

def fp_d2s(x, K = 8, elim = 4, w = 1, alpha = 20):
    if x > alpha:
        x = alpha

    man_start = alpha
    exponent = 0

```

```
    while man_start > x and exponent < 2**elim - 1:
        exponent += 1
        man_start = man_start / 2

    T = man_start * 2* 2**(-K) * np.array([float(2 ** (K - i))
        for i in range(1, K + 1)]).astype(np.float32)

    exponent = format(abs(exponent), "b")

    while len(exponent) < elim:
        exponent = "0" + exponent

    # mantessa
    z = []
    for t in T:
        if t <= x:
            x -= t
            z.append("1")
        else:
            z.append("0")

    man = ''.join(z)

    return exponent + man
```

Listing 2: Python Code for Floating-point neuron

One issue that presents itself when trying to code a floating-point neuron is the time step coding requirement. With a Few Spikes neuron, the code is run at each time step and an output is generated with each time step. However, with a floating-point neuron, the production of the exponent and the mantissa are done in one time step when the final spike is received, this issue will need to be resolved before being implemented into a Spiking Neural Networks.

7.1.6 Primacy Code

Humans use five major senses: hearing, sight, taste, touch and smell, it is the latter that primacy coding looks into. The senses from the olfactory system are processed regardless of concentration on the source. This ability for odor detection to be done at this level suggests an early odor-evoked neural activity [20]. It is this neural activity that will be investigated to see if there is a potential for a new method of neural coding that could be implemented.

7.1.7 Spiking Neural Network training databases

There are three main data sets to be used across this project for training the Spiking Neural Networks, they range from relatively small, allowing for a relatively simple neural model to be trained, to relatively complex data sets which will require more complex models.

MNIST [7] (Modified National Institute of Standards and Technology database) is a data set containing 60,000 training and 10,000 testing images of various handwritten digits. The images are size-normalised, centered and have the relatively small size of 28 by 28. The relatively small size of these images makes it ideal for training small Spiking Neural Networks on due to the relatively small input layer needed.

CIFAR-10 [8] is a database consisting of 60,000 32x32 colour images that can be classified into 10 classes with 6,000 images per class. This is a small jump up from MNIST so will allow for a larger training network.

Omniglot[9] is a more complex version of MNIST. MNIST only contains 26 different alphabetical symbols in different styles. Omniglot in comparison contains 340 differnt writing styles, including Abjads, Alphabets, Abugidas, Syllabaries and Semanto-phonetic scripts[21]. Whilst MNIST has 60,000 training data, Omniglot has far less, so training on this data set offers a different challenge.

7.2 Resources Required

This project is primarily programming focused, therefore the hardware resources needed is access to a computer which can run GeNN and Python. Unfortunately, the University computers do not

allow GeNN to run, so most of the light computational work will be done on a personal laptop, any larger and heavier computational work can be done with a personal desktop equipped with an GTX 1050TI, a GPU who's performance has been measured as okay by PyGeNN [22]. If more computational processing is required, then there is access to GPU clusters within the GeNN group for the training of larger models.

The software requirements involve the use of GeNN (GPU enhanced Neural Networks) which is a University developed software at Sussex, support for this software is readily available with the developers still being at the University of Sussex.

8 Methodology

8.1 Theoretical Implementation

Due to the complexities of implementing a new neural coding method into GeNN directly, it is more sensible to develop and test these models first with a theoretical model before implementing them into GeNN. This introduces two advantages: the first is the ability to test the performance of this model on datasets to gain a hypothesis as to how these models may perform in the event of the implementation not being successful, the second advantage is being able to test that these models are possible and to improve the efficiency before implementing them into the GeNN framework.

Being able to focus on the mathematical representation of each model allows for a greater understanding of how it would need to be implemented into GeNN, as well as a greater understanding of how the model could be later improved.

The theoretical approaches would be completed within python using expected python libraries. To compare the models, the precision and range of the value representation would be tested, as well as viewing the range of values between the datasets to be tested to predict which model would perform best in each dataset.

8.1.1 Few-Spikes

The Few-Spikes model had already been implemented into GeNN, however, for the theoretical comparison between the two approaches, a simplified model will be implemented to convert a rational number into a spike train.

The code for a Few-Spikes model was already discussed in the background research section 7.1.4, however, there were some minor changes needed before implementing this version into GeNN. The code previously discussed relies on a loop to iterate over the input value and to generate the spike train, within a neuron however, this would not be optimal. Instead, it is important that at each time step, the input value is assessed and an output spike needs to be determined. Therefore, the revised code and Few-Spikes model would need to be altered and more close to the code found in figure 3.

```
# Few Spikes Neuron
# x = input
# K = timesteps K = 1, ... t
# alpha = value representation range (cap)

import numpy as np

def fs(x, K = 10, alpha = 25):
    t = 0
    fx = 0
    v = np.zeros(K)
    z = np.zeros(K)

    T = alpha * 2**(-K) * np.array([float(2 ** (K - i))
                                    for i in range(1, K + 1)]).
    astype(np.float32)
    h = alpha * 2**(-K) * np.array([float(2 ** (K - i))
                                    for i in range(1, K + 1)]).
    astype(np.float32)

    v[0] = x
```

```
20
21      while t < K:
22          # spike if voltage > threshold, reset if spike.
23          if v[t] >= T[t]:
24              z[t] = 1
25              v[t] = v[t] - h[t]
26
27          # copy over value once reduced.
28          if t + 1 < K:
29              v[t + 1] =  v[t]
30
31          # sum voltage if spike
32          fx += z[t] * h[t] #* 0.5
33          t += 1
34
35      return z, fx # outputs tuple of spike train and sum (sum for
    configuration)
```

Listing 3: Python Code for Floating-point neuron

8.1.2 Floating-Point

To get a model to compare the differences between FS and FP, the structure discussed in section 7.1.5 would be suitable for a direct comparison, however, for an implementation into GeNN there would need to be some changes to the code.

The current method relies on a loop to iterate through in order to generate the exponent value and where the mantissa needs to start. This is where the first change needs to be made for compatibility within GeNN. As the code will be be implemented into a system which relies on code being ran at each timestep, the code would need to accommodate accordingly.

Therefore, the updated code as seen in figure 4 allows for the generation of the mantissa start (man_start) as well as the exponent value without the need to iterate. This will allow these two values to be calculated at the first timestep.

```
1  # Floating-point Neuron
2  # x = input
3  # K = timesteps K = 1, ... t
4  # elim = length of exponent
5  # w = weight on neuron
6  # alpha = value representation range (cap)
7
8  import numpy as np
9
10 alpha = 20
11
12 def fp_d2s(x, K = 8, elim = 4, w = 1, alpha = 20):
13     # current timestep
14     t = 0
15     # membrane voltage
16     v = np.zeros(K + elim)
17     # output spike train
18     z = np.zeros(K + elim)
19
20     # starting value for the mantessa scale
21     man_start = alpha * 2** - min((2**elim - 1), max(0, math.ceil(
    math.log2(1 /(x / alpha)))))
22     print(man_start)
23     # the integer exponent value
24     exponent = min((2**elim - 1), max(0, math.ceil(math.log2(1 /(x
    / alpha)))))
25     print(exponent, elim)
26     # mantessa scale, starting at mantessa start
```

```
    m = man_start * 2* 2**(-K) * np.array([float(2 ** (K - i)) for i in range(1, K + 1)]).astype(np.float32)
    # exponent scale, dependend on length of elim
    e = 16 * 2**(-elim) * np.array([float(2 ** (elim - i)) for i in range(1, elim + 1)]).astype(np.float32)

    # placing the values of exponent and mantessa into output train
    v[0] = exponent
    v[elim] = x


    while t < K + elim:
        #for calculating exponent
        if t < elim:
            if v[t] >= e[t]:
                z[t] = 1
                v[t] = v[t] - e[t]

            # copy over value once reduced.
            if t + 1 < elim:
                v[t + 1] =  v[t] # no need to reduce further as this has already been shortened.

        #for calculating mantessa
        else:
            if v[t] >= m[t - elim]:
                z[t] = 1
                v[t] = v[t] - m[t - elim]

            # copy over value once reduced.
            if t + 1 < K + elim:
                v[t + 1] =  v[t] # no need to reduce further as this has already been shortened.

        t += 1


    return z
```

Listing 4: Python Code for Floating-point neuron

8.2 Practical Implementation

Comparing the results between the two approaches from a theoretical performance standpoint was first focused on the range and precision of the representation of values, and although this demonstrated the theoretical capabilities, a physical implementation is needed to test the model in a real world application.

As previously discussed in the Aims and Objectives section, there are 4 main stages to the practical implementation: implementing a Few-Spikes network to classify MNIST, implementing a Few-Spikes network to classify CIFAR-10, implementing a Floating-Point network to classify MNIST, and implementing a Floating-Point network to classify CIFAR-10.

Due to the complexities and differences between a typical artificial neural network and a spiking neural network, as well as the differences between the frameworks of TensorFlow and GeNN, developing these models on a small network before scaling them up to a larger network was the first step.

Two neurons will need to be developed for a few-spikes network; the first will be a neuron for the input later, this neuron will convert the input values passed through and return a spike train, the second type of neuron will be an inline neuron, which expects to receive spike trains, perform a function, then return the updated spike train.

To allow neurons to communicate between each other, they must be connected via a synapse. Within GeNN, this is done by adding a synapse population with a weight update model.

Therefore, to develop and implement these two approaches, three sets of classes will need to be created: a custom neuron class for the input neuron, a custom neuron class for the inline neuron, and a custom weight update class for the synapse population.

8.2.1 Developing a Few-Spikes Neuron

Input Neuron

As discussed before, the approach is split into three primary classes, the first to be developed is the input layer neurons. Within GeNN, a custom neuron is created using a custom neuron class, which will later be initialised and connected into a network. The code below in Listing 5 is the completed input neuron class, which has been developed by the GeNN Team [23].

```
fs_input_model = create_custom_neuron_class(
    'fs_relu_input',
    param_names=['K', 'alpha', 'elim'], #k = timesteps, alpha = highest value, elim = length of exponent
    derived_params=[("scale",
                     create_dpf_class(lambda pars, dt: pars[1] * 2**(-pars[0]))())],
    var_name_types=[('input', 'scalar',
                     VarAccess_READ_ONLY_DUPLICATE),
                     ('Vmem', 'scalar'),
                     ('scaleVal', 'scalar')],

    sim_code='''
    // Convert K to integer
    const int kInt = (int)$(K);

    // Get timestep within presentation | mod to keep within range
    const int pipeTimestep = (int)($(t) / DT);

    // If this is the first timestep, apply input
    if(pipeTimestep == 0) {
        $(Vmem) = $(input);
    }

    // check pipetimestep < 8
    const scalar hT = $(scale) * (1 << (kInt - ((pipeTimestep % kInt)+1)));

    $(scaleVal) = $(scale) * (1 << (kInt - ((pipeTimestep % kInt)+1)));
    ''',
    threshold_condition_code='''
    $(Vmem) >= hT
    ''',
    reset_code='''
    $(Vmem) -= hT;
    ''',
    is_auto_refractory_required=False)
```

Listing 5: Custom Neuron Class for FS Input Neuron

The first parameter passed is the name of the class for identification.

The second parameter is static parameter values which will be passed through the class, these are static values which won't change at each iteration, these values include the length of K, and the alpha value for this neuron / network.

The third value to be passed through is the derived parameters, as the name suggests, this is a value that is calculated given the static parameters passed through. For this class, the derived parameter is the scale from which a spike is determined, for few-spikes, this is the h value discussed before.

The forth parameter is the variable values, which can change at each timestep. The three used are: the input value which is the rational value which will get converted to a spike train, the voltage membrane value (Vmem) which will be set to 0 initially, and finally the scale value (scaleVal) which will also be set to 0 initially.

The fifth parameter is the simulation code, this is the code which will get run at every timestep and where the bulk of the neurons functionality comes from. The simulation code checks if the

current iteration is at the start of a spike train / pipeline, if this is true, then the membrane voltage (Vmem) is set to the input value. The h value (also referred to as h(t), or in the code as hT or scaleVal) is the halved alpha value at each timestep, this is the value which will be compared with later to determine if a spike condition has been met.

The sixth parameter is the threshold condition code, this is checked at every time step to see if conditions are met, for this model, the condition is if the voltage membrane (Vmem) is greater or equal to the scaled h value. If the conditions are met, then a spike is registered.

The seventh parameter is the reset code, which is ran once the threshold condition code is met, for this model, the reset code will reduce the voltage membrane value by the h value.

Inter Neuron

After the input layer of neurons have converted the signal into spikes, these need to feed into an inter neuron, this neuron will take a spike as an input, where after being manipulated, they will be sent downstream. The code below in Listing 6 is the completed inter neuron class, which was developed by the GeNN Team [24].

```
fs_model = create_custom_neuron_class(
    'fs_relu',
    param_names=['K', 'alpha', 'upstreamAlpha'],
    derived_params=[("scale", create_dpf_class(lambda pars, dt:
   pars[1] * 2**(-pars[0]))()),
                    ("upstreamScale", create_dpf_class(lambda pars,
    dt: pars[2] * 2**(-pars[0]))())],
    var_name_types=[('Fx', 'scalar'), ('Vmem', 'scalar')],
    sim_code='''
    // Convert K to integer
    const int kInt = (int)$(K);

    // Get timestep within presentation
    const int pipeTimestep = (int)($(t) / DT);

    // Calculate magic constants. For RelU hT=h=T
    // **NOTE** d uses last timestep as that was when spike was
   SENT
    const scalar hT = $(scale) * (1 << (kInt - (1 + (pipeTimestep %
    kInt))));
    const scalar d = $(upstreamScale) * (1 << ((kInt - pipeTimestep
   ) % kInt));

    // Accumulate input
    // **NOTE** needs to be before applying input as spikes from
   LAST timestep must be processed
    $(Fx) += ($(Isyn) * d);

    // If this is the first timestep, apply input
    if(pipeTimestep == 0) {
        $(Vmem) = $(Fx);
        $(Fx) = 0.0;
    }
    ''',
    threshold_condition_code='''
    $(Vmem) >= hT
    ''',
    reset_code='''
    $(Vmem) -= hT;
    ''',
    is_auto_refractory_required=False)
```

Listing 6: Custom Neuron Class for FS Inter Neuron

The inter neuron class shares a lot of similarities to the previous input neuron seen in listing 5. The first significant difference is the variable values that are passed through, Vmem is still passed

but Fx is a new parameter which accumulates the spikes from the previous pipeline until it is ready to be processed. Once all the spikes from the previous pipeline have been accumulated, it will get copied over to Vmem where Fx will be reset.

The fifth parameter, the sim code, has a significant difference from the previous neuron. Fx accumulates the signals from the previous neuron as seen in line 21 in Listing 5 where the signals from input to the neuron are acquired using the isyn value.

At pipe timestep 0, the accumulated signals from the previous pipeline which are now represented by the variable Fx are copied over to the voltage membrane, with Fx reset back to 0 for the next pipeline.

Threshold condition code and reset code remain the same for this neuron.

Synapse

To connect the neurons together into a larger network, synapses are used to carry the signal from one neuron to another.

For the FS neuron, a built in GeNN weight update model is used, titled *signed static pulse*. An example of the code is shown in Listing 7.

```
signed_static_pulse = create_custom_weight_update_class(
    'signed_static_pulse',
    var_name_types=[("g", "scalar", VarAccess_READ_ONLY)],
    sim_code='''
    $(addToInSyn, $(g));
    ''',
    event_code='''
    $(addToInSyn, -$(g));
    ''',
    event_threshold_condition_code='''
    $(input_pre) < 0.0 && spike
    '''
)
```

Listing 7: Custom Weight update Class for FS Synapse

This weight update model / synapse, performs a basic function of moving spikes from one neuron to another without any manipulation or modification to the spikes. This works well for the FS network as the processing is completed within the neurons and each neuron / spike train shares the same K and alpha value. For the FP network, due to the variety of K and alpha values around the network, the synapse will take on a greater role to allow compatibility between neurons / spike trains with different k and alpha values.

8.2.2 Implementing a Few-Spikes network to classify MNIST

As Few-Spikes conversion is a method of converting a pre-trained neural network into a Spiking Neural Network, a pre-trained model needs to be created.

As MNIST database challenges a network's computer vision capabilities, convolutional layers in addition to fully connected layer were used. Figure 3 provides an illustration to the architecture of the network used for this computer vision task.

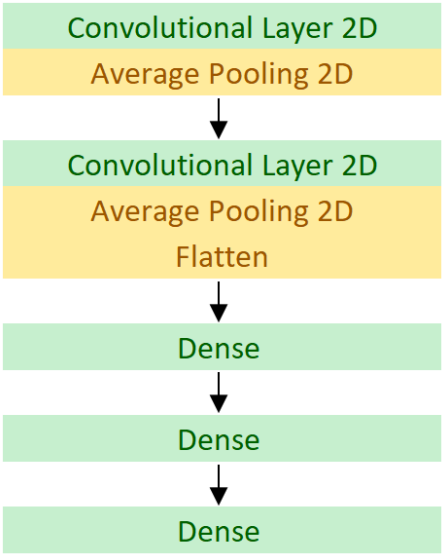

Figure 3: MNIST Neural Network Architecture

As MNIST is a dataset consisting of images with relatively small dimensions, a large neural network isn't necessary in order to solve this problem, therefore, a neural network with 5 layers was chosen. As seen in Figure 3, the network consists of 2 convolutional layers with both having average pooling, pooling allows for reducing the spatial dimensions of the layers to aid in identifying features.

After these two convolutional layers, are three fully connected layers which can use the features extracted from the previous two layers.

Once the model has been ran and trained on the dataset, it can then be converted to a Spiking neural network through GeNN. First a converter is created, for Few-Spikes conversion, this will be a Few-Spikes converter. The TensorFlow model that has been trained is then passed through GeNN's *convert_tf_model* class along with the converter which was created. The new converted SNN model can then be evaluated and the accuracy can be tested.

8.2.3 Implementing a Few-Spikes network to classify CIFAR-10

Implementing Few-Spikes to solve a larger dataset such as CIFAR-10 required some minor changes. The network designed to solve the MNIST dataset was a relatively small network with only 5 layers of depth. To solve a more demanding dataset where the dimensions of the images are greater along with more channels for colour, requires a deeper neural network.

Therefore, to create a classification neural network for the CIFAR-10 dataset a network consisting of 10 layers was chosen as seen in Figure 4. The size of this network can range drastically, with networks significantly smaller generating significant results. A network with a depth of 10 was chosen as a compromise between time required for training and performance. The design for this network is inspired from the VGG16[25] network, with modifications required to allow for compatibility with GeNN, as well as a reduction in size to allow for improved training times.

Once a TensorFlow neural network was created and trained, it would be passed through GeNN's *convert_tf_model* with the results evaluated and tested.

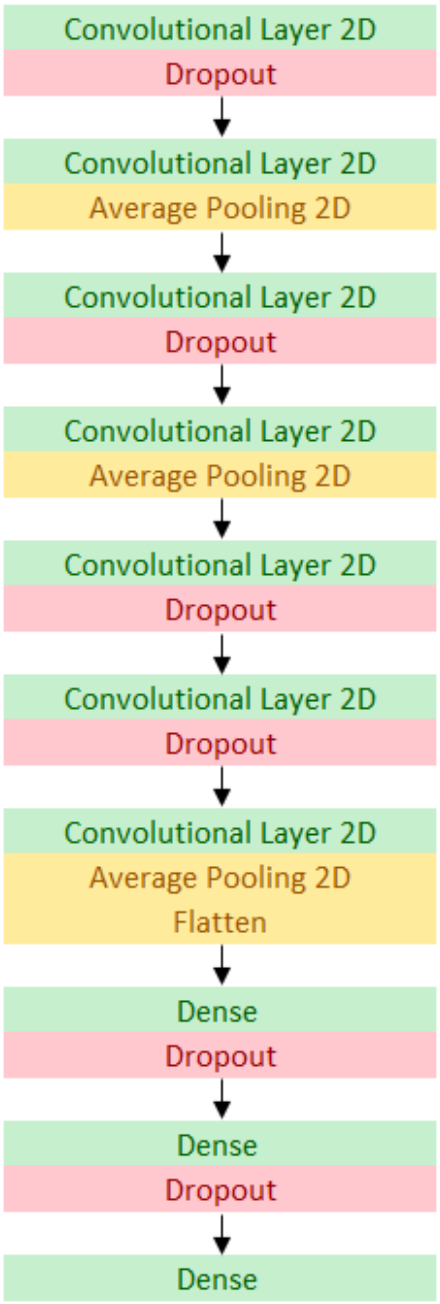

Figure 4: CIFAR-10 Neural Network Architecture

8.2.4 Developing a Floating-Point Neuron

Developing the Floating-Point neuron within GeNN followed a lot of the same classes, with *create_custom_neuron_class* being the class used to create the custom neuron and the primary focus of this approaches development.

However, a successful development of the Floating-Point neuron was never successfully implemented. The code and classes discussed will look into how an Input neuron was developed and later tested, however, an Inter Neuron and appropriate weight update model / synapse was never successfully developed.

A discussion at the end of this report will look into why a successful development was not completed, along with suggestions as to how an implementation could be achieved in future efforts. Theoretical testing results therefore will be the best benchmark as to how these two approaches perform.

Input Neuron

The Input Neuron converts the input rational number into a spike train. For the Floating-Point neuron, this will involve sending two sets of information within one spike train: an exponent value, then a mantissa value. The number of *steps* within the spike train which are for the exponent will be a parameter which is passed through when initialising the neuron. Within the network, this value, which will be referred to as elim (the number of values which will represent the exponent), will remain the same for all neurons and only adjustable when initialising the network.

The parameters for the Floating-Point neuron share a lot of similarities with the Few-Spikes input neuron, Figure 5 shows the code for an approach at a Floating-Point input neuron.

Four main differences are the parameters passed, the lack of derived parameters (in this example), the variable parameters and the simulation code.

There is an additional parameter passed: *elim*. As discussed before, *elim* is the number of steps within the spike train which will carry the exponent value.

A lack of derived parameters is not currently used as it required the use of the variable parameters rather than the fixed parameters, which made it easier to calculate this value within the simulation code instead.

Variable parameters included an extra parameter: exponent. This value represents the exponent value which will be calculated from the alpha, input and elim values. In later iterations of this Inout neuron, this value could be removed as a parameter.

One final significance lies within the simulation code. Unlike Few-Spikes, the Floating-Point approach is more dynamic with changing alpha values from the exponent, different scale and different values making up the spike train.

The simulation code can be split into two sections: The exponent, and the mantissa. To calculate the exponent value, the following equation is used:

$$min((2^{elim-1}), max(0, ceil(\log_2(\frac{1}{\frac{input}{\alpha}}))))$$

This equation is determining the opimal number of divides than the alpha value needs to have, given the input value

To calculate the start of the mantissa scale, the following equation is used:

$$\alpha \cdot 2^{-min((2^{elim-1}), max(0, ceil(\log_2(\frac{1}{\frac{input}{\alpha}}))))}$$

The mantissa scale starts, based on the exponent value. I.e. If the input value is 7, and the alpha value is 20, then the exponent will be 2 as 20 will be halved twice down to 5 (first to 10, then to 5). Therefore, the exponent will be 2, whilst the mantissa start will be 5.

With these two values found, the simulation code then will spend a spike when conditions are met whilst the pipe timestep is below the elim value, this is to generate the exponent spike train. Once the pipe timestep is greater than the elim, then the mantissa is calculated and the mantissa spike train is generated.

```
1 fp_relu_input_model = create_custom_neuron_class(
2     'fp_relu_input',
3     param_names=['K', 'alpha', 'elim'], #k = timesteps, alpha =
    highest value, elim = length of exponent
4     var_name_types=[('input',   'scalar',
    VarAccess_READ_ONLY_DUPLICATE),
```

```
                    ('Vmem',     'scalar'),
                    ('scaleVal','scalar'),
                    ('exponent','scalar'),
                    ('hT',       'scalar')],
    sim_code='''
    // Convert K to integer
    const int kInt = (int)$(K);

    // Convert Alpha to integer
    const int AlphaInt = (int)$(alpha);

    // Convert elim to integer
    const int elimInt = (int)$(elim);

    // Get timestep within presentation
    const int pipeTimestep = (int)($(t) / DT);

    // If this is the first timestep, apply input
    if(pipeTimestep == 0) {

        $(scaleVal) = AlphaInt * pow(2, - fmin(pow(2, elimInt - 1), fmaxf(0, ceil(log2(1 / ($(input) / AlphaInt))))));

        // scaleVal can be derived from exponent
        $(exponent) = fmin(pow(2, elimInt - 1), fmaxf(0, ceil(log2(1 / ($(input) / AlphaInt)))));

        $(Vmem) = $(exponent);

    }

    if (pipeTimestep == elimInt) {
        $(Vmem) = $(input);
    }

    $(hT) = 0;

    $(measure) = 0;

    if (pipeTimestep >= elimInt) {

        $(hT) = $(scaleVal) / (1 << (pipeTimestep - 4));

    } else {

        $(hT) = pow(2, elimInt-1) / (1 << pipeTimestep);
    }
    ''',
    threshold_condition_code='''
    $(Vmem) >= $(hT)
    ''',
    reset_code='''
    $(Vmem) -= $(hT);
    ''',
    is_auto_refractory_required=False)
```

Listing 8: Custom Neuron Class for FP Input Neuron

9 Experiments

Experiments are split into two sections: Theoretical and Practical. Due to the limited successes on the Floating-Point neuron, looking at the theoretical results allows for some conclusions to be drawn about this approach.

9.1 Theoretical Results

As discussed in previous sections, testing of the models and different neuron types was first developed theoretically to be tested and improved upon before being implemented into GeNN.

Two areas were tested: Range and precision of the neurons and the range of values found within the datasets. With these two pieces of information, a greater understanding of how the floating-point approach may compare against the few-spikes method.

9.1.1 Range and Precision of FS and FP Neuron

To test the range and precision of the two approaches, each approach will have a range of values passed through to be encoded. These encoded spikes will then be converted back to a rational value, and the difference between the two compared.

A variety of different ranges, alpha values and configurations of floating-point will be tested to determine the strengths and weaknesses of both approaches.

To determine which approach is better performing, a percentage difference will be used. The percentage difference is calculated by accumulating the difference between the value inputted, and the output value after conversion. The closer the difference, the better the model is performing. A percentage difference between the two approaches is the primary metric to compare the two. A negative percentage difference (% difference) indicates a better performance from the Few-Spikes approach.

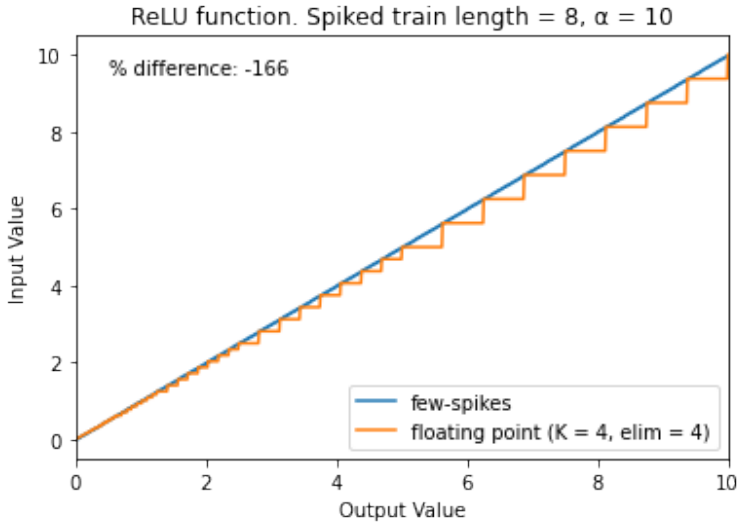

Figure 5: Value representation of FS and FP neurons with a range of 0 to 10, K = 8 and Alpha = 10

The first comparison between the two approaches was comparing the value representation of 1000 rational values between 0 and 10. For this test, the length of the spike train was set to 8 for both approaches, with an alpha value of 10 also being shared. As for the composition of the floating-point representation spike train, the first test was completed with 4 bits representing the mantissa, whilst 4 bits were used to represent the exponent.

The results from this first test indicated that the performance of the floating-point neuron is outperformed by the few-spikes neuron. However, this is with the current configuration of the floating-point neuron, and the assumption that the significance of the range of values is linear.

Plotting the results from this first test as seen in figure 5 gives a better picture as to how the two approaches perform. Floating-point neuron showed a deteriorating performance with larger values, indicating that the size of the mantissa potentially provided a bottleneck in the precision of the larger values.

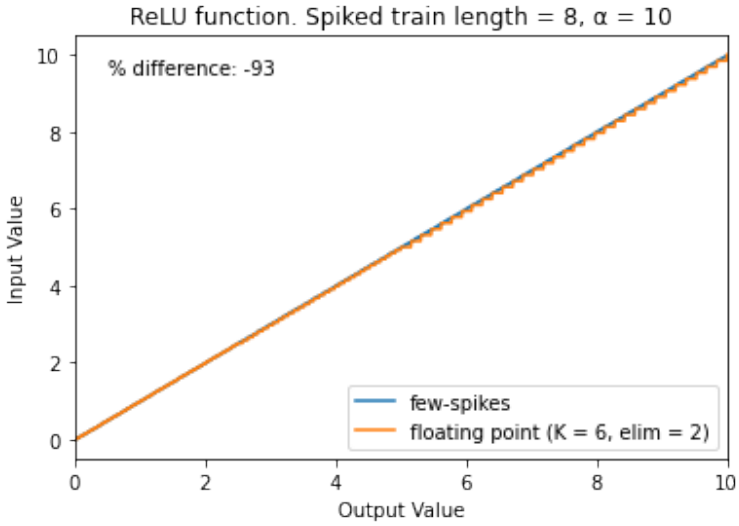

Figure 6: Value representation of FS and FP neurons with a range of 0 to 10, K = 8 and Alpha = 10. Floating point is made up of a 6 spike mantissa and a 2 spike exponent)

To test the different configuration options which floating-point representation can have, a larger number of bits was used to represent the mantissa, with a reduced number of exponent bits. For the next test, the mantissa will be comprised of 6 bits, whilst the exponent will be tested with only 2 bits, still totalling 8 bits overall. As seen in figure 6, this did improve the performance of the FP neuron, however, this was still under performing compared to the FS model.

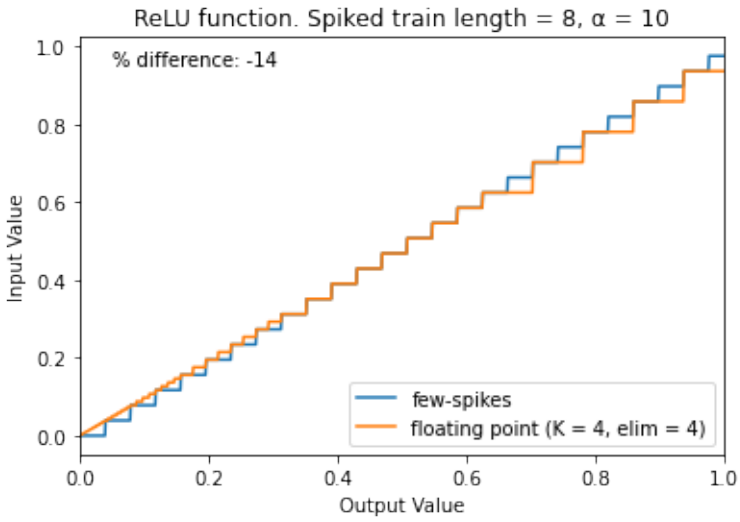

Figure 7: Value representation of FS and FP neurons with a range of 0 to 1, K = 8 and Alpha = 10

The next test was looking at a smaller range with a large alpha. As seen in earlier tests, the floating-point method out performed the few-spikes approach when it came to representation of smaller values, however it didn't perform overall greater. The next test would look at a range of 0 to 1 with an alpha value of 10.

The results from looking into the smaller values showed that floating-point does perform greater at lower values.

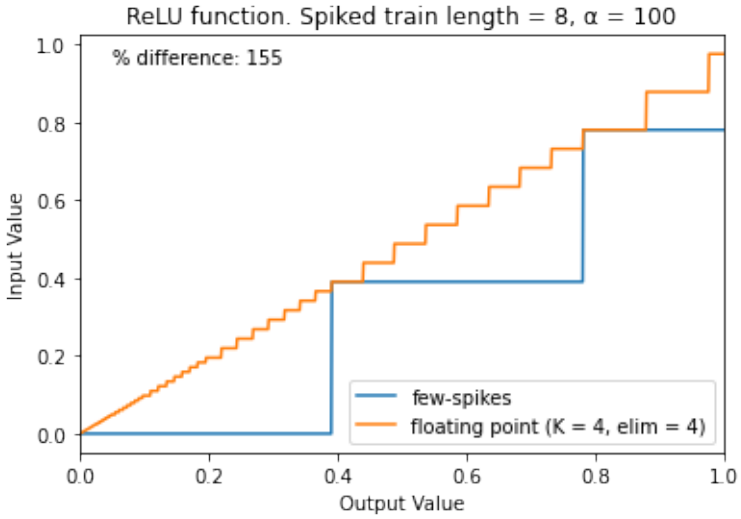

Figure 8: Value representation of FS and FP neurons with a range of 0 to 1, K = 8 and Alpha = 100

This was confirmed again when increasing the alpha value further as seen in figure 8. This suggests a use case when requiring the accuracy of smaller values to be more significant. However, overall, few-spikes still outperforms floating-point representation. At least with the current configuration.

The final set of tests was following the trend where a larger mantissa returned a greater performance. With 8 being to max total number of bits, a mantissa of 7 was chosen, allowing for a 1 bit exponent.

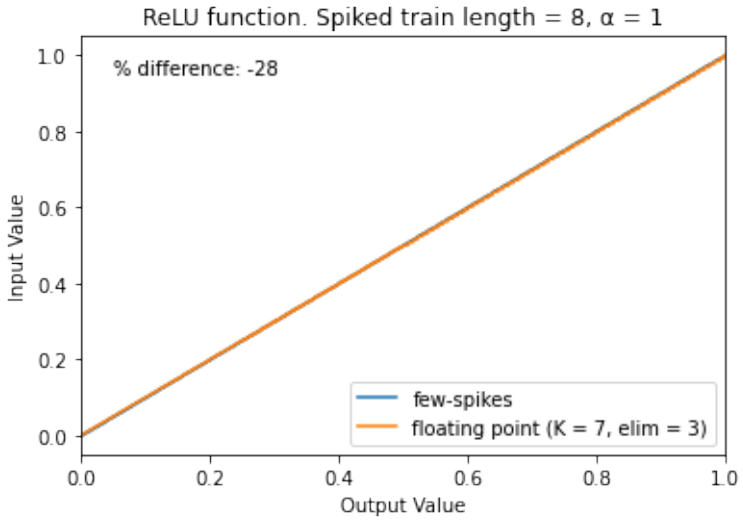

Figure 9: Observing the impact of floating-point neuron with different elim and mantissa bits, with a range from 0 to 1

The results in figure 9 show that the performance of floating-point neuron is only marginally beaten by that of the few-spikes neuron.

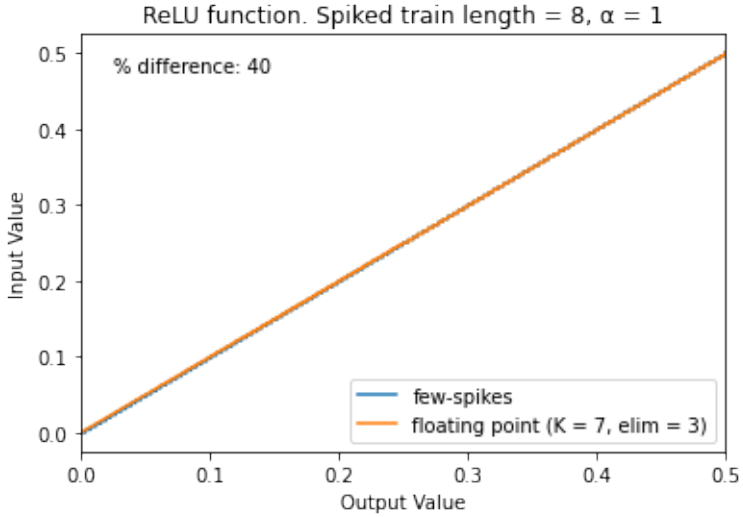

Figure 10: Observing the impact of floating-point neuron with different elim and mantissa bits, with a range from 0 to 0.5

A closer comparison, especially when just representing the smaller values such as the range 0 to 0.5 with an alpha of 1, shows that floating-point neuron outperforms few-spikes.

This trend continues with figure 11 showing that the smaller values again, this time between the range of 0 to 0.25 with an alpha value of 1, are greatly better represented compared to few-spikes.

The results from this theoretical testing therefore suggest that for floating-point representation, a larger mantissa is advantageous for a better representation. The results have also highlighted that the model would do well in a dataset where the majority of the values are skewed to the smaller size, rather than just linear where few-spikes is more appropriate.

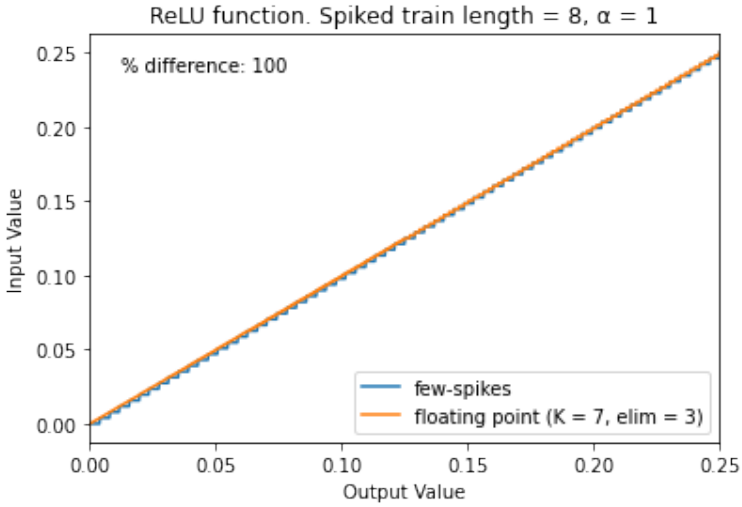

Figure 11: Observing the impact of floating-point neuron with different elim and mantissa bits, with a range from 0 to 0.25

9.1.2 Dataset value range

With the results gathered from the value representation, floating-point representation performed favorably when the spread of values was skewed positively so the majority of the values were smaller. Observing the spread of values across the different datasets to be tested would allow for a prediction to be made on how the approach may perform.

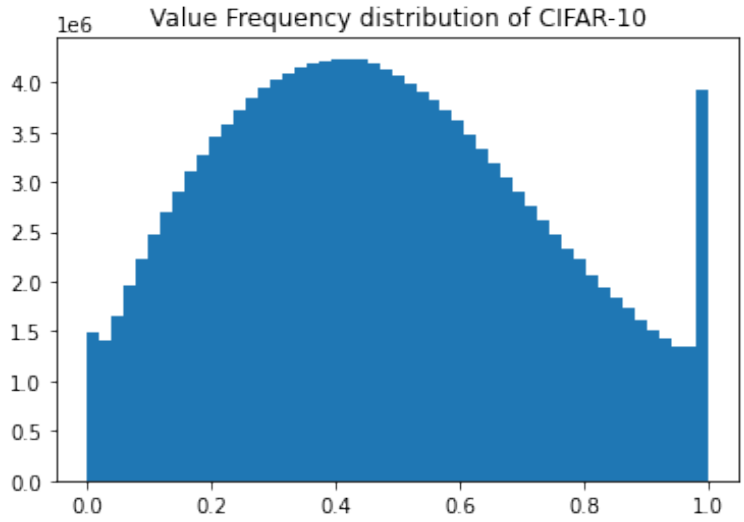

Figure 12: Dataset value frequency distribution

As seen in figure 12, the spread of data is slightly positively skewed, which would favor the floating-point approach. The average value across all channels and images comes to 0.473 (3.s.f) with a standard deviation of 0.251 (3.s.f). Although this is favourable, the values are still relatively evenly spaced, so although it would seem that floating-point may have an advantage theoretically, practically there is still a chance of this approach not being the best performing.

9.2 Practical Results

Practical results involved creating a small network of neurons with an input neuron and a couple of inter-neurons, then testing the neurons with larger networks on predetermined datasets.

Due to the limited success with the Floating-Point neuron, the main focus will be on the Few-Spikes neuron, with the results from the Floating-Point being the firing of the input neuron.

9.2.1 Few-Spikes Model

Small Network

The first stage was to implement a small network consisting of 3 neurons with an input neuron, and 2 connected neurons. The preliminary results from this implementation would allow for testing of the neuron to ensure it is behaving as expected.

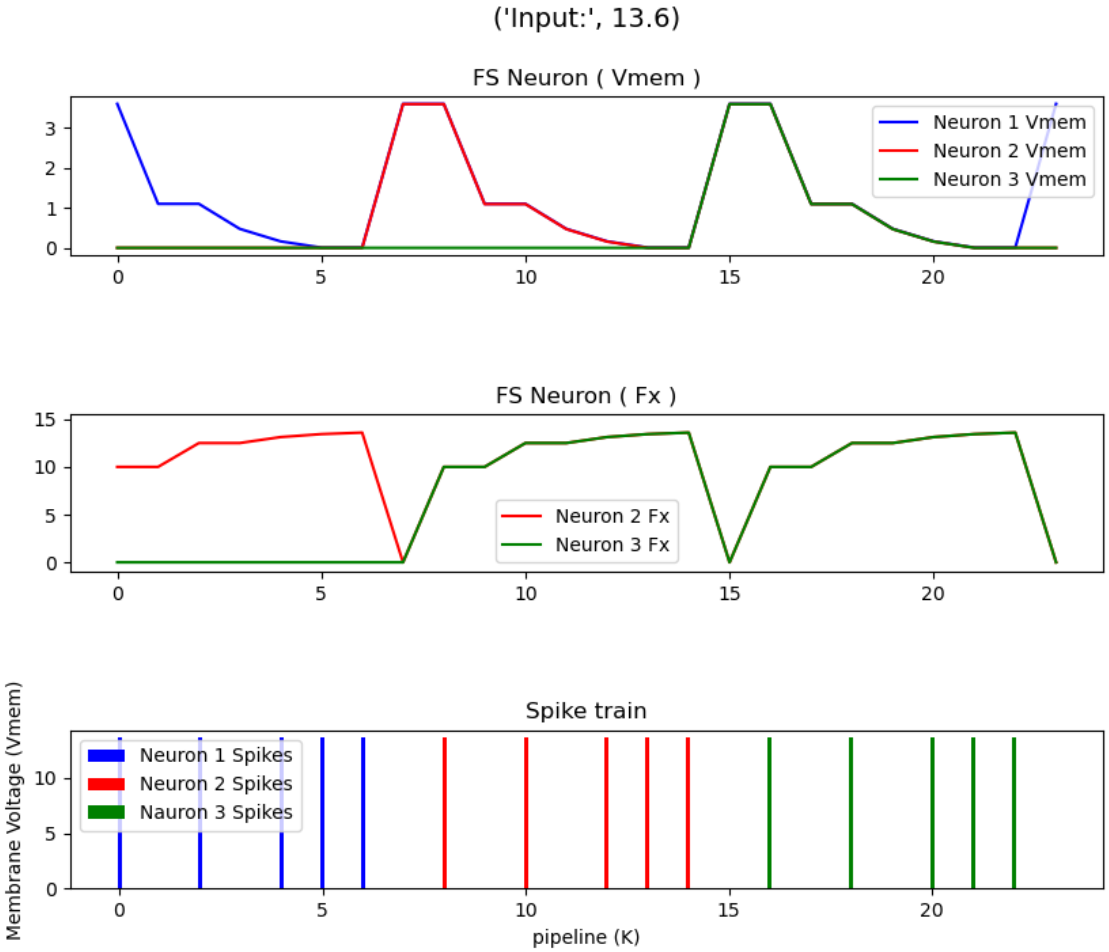

Figure 13: Few-Spikes network consisting of 3 neurons

Figure 13 shows the changing values of three connected neurons. The top chart shows the membrane voltage over time for the three neurons, note how when the voltage decreases, the Fx value of the following neuron (as seen in the middle chart) increases proportionally. Along with this, there is a spike (as seen by the bottom chart) when the voltage decreases and Fx increases as this is from the condition code being met within the neuron class. The bottom chart shows the output spike train from each neuron.

MNIST

The MNIST implementation was successful in part due to the process already being completed by the GeNN team. There are two stages of results: the evaluation of the TensorFlow network, and the evaluation of the Few-Spikes converted network.

The results in the table below show how the converted network performed compared to the TensorFlow model.

Network	Accuracy
TensorFlow	99.01%
Few-Spikes Converted Network	99.01%

Due to the extremely high accuracy scores and low loss value (0.0187), the model appears to be performing well. The closeness between the TensorFlow model and the converted Few-Spikes model also demonstrates that the Few-Spikes method is able to represent values to an appropriate level where the accuracy is maintained.

CIFAR-10

Similar to the results section for MNIST, CIFAR-10 was a straight forward evaluation where a model is first trained to create a TensorFlow model, then converted to a Spiking Neural Network.

Due to the size and complexity of the images that CIFAR-10 offers compared to that of MNIST, the time to train and convert this network was significantly longer. An epoch size of 10 was chosen for both networks.

Network	Accuracy
TensorFlow	52.34%
Few-Spikes Converted Network	52.06%

The results from the CIFAR-10 neural network demonstrate that Few-Spikes is able to successfully maintain the performance of the original network. Although the performance is not as great as the results from the MNIST neural network, this will be down to the training time and the size of the network.

9.2.2 Floating-Point Model

Due to the lack of success in completing a Floating-Point implementation, the only results presentable are the properties of the Input neuron over time.

Similar to the results shown by the Few-Spikes neuron as seen in Figure 13, a single neuron's performance will be observed in Figure 14.

Input Neuron

As discussed in the Methodology section, the behaviour of the Floating-Point neuron is comprised of two values being carried: the exponent and mantissa. Figure 14 shows the behaviour of an Input Floating-Point neuron when an input value of 7.23 is passed through.

As the alpha value used in this example is 20, the exponent value will be 2 (Due to 20 being halved twice to make the *new* alpha value become 5). The exponent value is sent first and so 2 is represented in binary as a 1 followed by a 0. This spike behaviour can be observed in Figure 15 with the first timestep being the 1, and the second timestep being the 0.

Figure 14 has a dotted line at X = 2, this is to illustrate *elim* and therefore any spikes before that line are for the exponent, whilst any spikes post the line are representing the mantissa. Figure 14 then shares similar properties to the Few-Spikes neuron, where when the Vmem value decreases, a spike is produced.

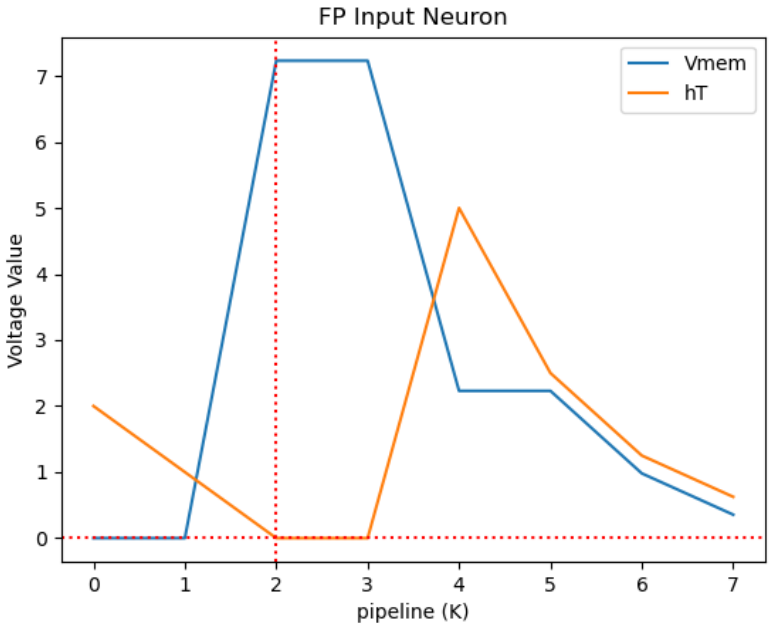

Figure 14: A Floating-Point Neuron with an input value of 7.23, elim of 2, Alpha of 20 and a K of 8

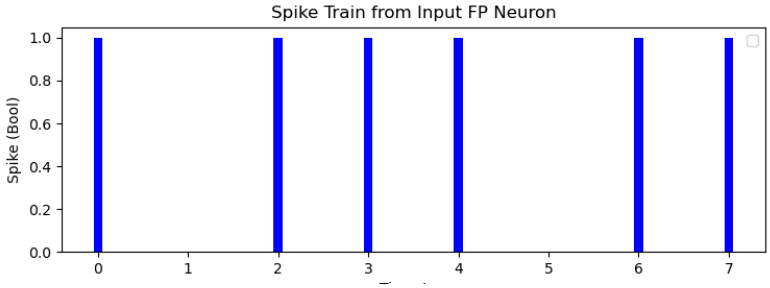

Figure 15: Spike train from FP neuron with an input of 7.23

10 Results

Due to the incomplete practical implementation of the Floating-Point approach, a comparison of the Floating-Point approach against the Few-Spikes approach will primarily be reviewed from a theoretical perspective.

When looking into the accuracy in value representation between the two approaches in section 9.1.1, the two approaches demonstrated performance advantages within certain parameters. Few-Spikes was able to demonstrate a consistent level of performance linearly due to the fixed alpha value not impacting the performance under certain areas. This approach and linear performance is optimal when a dataset consists of an even distribution of values.

The approach being investigated by this report, Floating-Point, was unable to match the performance of the Few-Spikes neuron when attempting to represent an even distribution of values whilst treating their significance equally. However, where the Floating-Point approach was able to outperform that of the Few-Spikes approach was when the frequency of smaller values was greater, and therefore being able to represent these values more accurately was advantageous. The reason behind this advantage is due to the exponent allowing for the mantissa to start at a much smaller value, however, with bits removed from the mantissa, when attempting to represent larger values, the lack of mantissa bits significantly reduces the capacity.

Looking into the distribution of values across different datasets, especially the datasets on which these approaches will be tested on, will help in developing a hypothesis as to which approach would perform best when implemented into a practical model. The first dataset value distribution to be observed was the MNIST dataset. Due to the MNIST dataset representing digits and not more complex shapes, the distribution of values fell either in 0 or 1, with an insignificant number of values between these. Without a relatively wide distribution of values, neither approach can be predicted to outperform the other, which indicates the likely possibility that both approaches will perform equally on this dataset. The next dataset to be observed was the CIFAR-10 dataset. This dataset offers far greater complexities relative to MNIST with a wide range of features to be extracted across 3 colour channels. This added complexity and additional colours generates a wider range of values, which unlike MNIST, creates a more complex distribution. Observing this distribution was seen in Figure 12 where the positive skew of values appeared to make the Floating-Point approach advantageous for this dataset.

However, theoretical testing is only useful in generating a prediction. Placing the two approaches against each other and implementing them into a network would generate the most reliable results. Unfortunately, this was not completed successfully for reasons which will be discussed in the next section. Therefore, looking into the theoretical results, it would be fair to assume that the performance between both approaches would be close.

11 Discussion and Conclusion

The primary goal and focus of this project was the implementation of the Floating-Point neuron into a neural network to solve complex machine vision tasks, and although this aim was not met, significant research, lessons, and understandings were developed. Therefore, this discussion section is broken into three sections: what went well about this project? What didn't go well during this project? And what future work could be done if this project was to be continued?

Project Successes

This project was a continuation of research completed over the summer of 2021 into an alternative to an approach developed by Christoph Stöckl and Wolfgang Maass[6], the Few-Spikes

conversion. At the end of the summer research, an alternative approach was successfully developed: The Floating-Point neuron. However, this approach was developed within Python with features which wouldn't be compatible when attempting to implement into a neural network, this is where this project took over. Implementing this new approach required two tasks: the first was developing the approach so that it could be implemented into an event driven framework, and the second was understanding the GeNN framework and implementing the Floating-Point neuron approach into said framework.

What went well was the successful completion of the first task and a partial completion of the second. The code from the summer research relied on loops to generate the exponent value and the mantissa start, something which wouldn't be practical to implement within GeNN. Instead, the loop was converted to a formula which would be ran at the first timestep. GeNN converts the code from the python classes into C to allow for faster and more efficient processing, therefore, making this formula compatible with C was of equal importance.

GeNN, as previously discussed, is a framework for generating Spiking Neural Networks which can run efficiently on GPUs to allow for the development and training of Spiking Neural Networks on conventional desktop hardware, instead of the typical alternative for Spiking Neural Networks which would be neuromorphic hardware or high power computer workstations. GeNN was a tool which was briefly introduced during the summer research, and so became a primary focus for this project. Although a successful GeNN implementation of the Floating-Point neuron into a network so far has not been achieved, understanding the structure of GeNN, how the classes interact, and observing how the Python code is converted into C, has greatly benefited my understanding of this framework, with the intention to further expand on my understanding of GeNN in future projects.

Project Failures

The clear answer to this, is the incomplete implementation into GeNN, however, there are several reasons to why this was not successfully done within this report.

The first is the complexities of the Floating-Point neuron compared to the Few-Spikes neuron. The Few-Spikes neuron uses a fixed alpha value which is shared across all neurons within the network. Although this may limit the range to which these neurons can represent values, it allows for easy connections between neurons. Within GeNN, the synapse which connects these neurons together simply passes on the signals from one neuron to another, so when the first spike is coming from different neurons, the first spike carries the same magnitude. Floating-Point alternatively, adds significant complications when creating a dynamic alpha value for each neuron, and for each spike train. Now, instead of all spikes coming into a neuron being treated equally, they must be adjusted and modified according to the exponent value, which also needs to be translated and interpreted.

This complication was attempted to be solved in this report by having the synapse / weight update model adjust the spike train before it arrived at the next neuron. The first attempt was to shift the spikes back or forward accordingly, however, after only a few layers, this would remove the advantages that Floating-Point had, which was the wide range availability. An alternative approach was to have a separate network carry the exponent value, which could then feed into the neuron so the incoming spikes could be adjusted accordingly, however, this would become a hybrid Spiking Neural Network, and reduce the efficiency of the network significantly. Alternative methods to solving this approach was discussed, however, future efforts on this project would benefit from experimenting with alternative solutions to this problem.

The second setback was the time allocated to learning and understanding GeNN. Although GeNN is made to be convenient to use through the functionality of coding in Python, it took longer than allocated to learn and understand. Although an improved understanding of GeNN is now achieved, future attempts should be done by first focusing on understanding how to implement a simple network within GeNN consisting of synapses and neurons.

Future Work

With the results produced from Spiking Neural Networks challenging that of conventional artificial neural networks, importance in researching this area of computational neuroscience is evident.

The advantage of converting already well developed artificial neural networks allows for significant time saving by not needing to train a network from scratch. This is the premise behind a lot of ANN-to-SNN converters, especially when training a Spiking Neural Network from the ground up is challenging. Continuing research into a Floating-Point conversion method remains a possibility, with alternative methods needing to be explored.

Although ANN-to-SNN converters do provide impressive results allowing for similar performance to the original model, the converted network is only encoding the ANN activations in the firing rate of the spiking neurons, the resulting SNN network using this conversion therefore re-

quires a relatively greater number of spikes to generate a similar performance, with the larger number of spikes outweighing the energy savings of the SNN. An alternative to converting a network is to train a SNN from the ground up. Although still a challenging task, there are emerging approaches which suggest positive results. A typical recurrent neural network will use a gradient descent learning technique called backpropagation though time, which unravels the network, allowing for both forward and backward passes. With a SNN, unravelling this network requires significant amounts of storage and processing. New approaches aim at reducing this demand by either looking into more efficient approaches, or alternative, more bio-inspired approaches. [26][27]

Conclusion Summary

So to conclude, although the main aim of this project was not accomplished, the aim for a project to facilitate in the developing of an understanding of a subject area, was successful. Understanding more about Spiking Neural Networks, the GeNN framework and neural encoding was all explored in depth across the duration of a year, with the motivation to continue and gain an even greater understanding.

Looking at the results, development and research, Few-Spikes, with its simplicity and linear value representation, is the greater approach of the two. Although in certain circumstances Floating-Point could prevail, the significance in difference between the two approaches is up for debate. The two approaches also fall under the same issue as discussed in the section above, where converting an ANN to a SNN potentially removes any cost savings that a SNN could offer.

12 Log

Date	Summary of Meet
16/09/2021	First meeting about FYP, talked about project outline, primary and secondary objectives and extension task. Will conduct research in the area of these goals and in the area of the project as a whole
29/09/2021	Meeting to discuss current progress, discussed project proposal form and converting the current version (in word) into Latex. Discussion into GeNN and understanding GeNN code
06/10/2021	went over the converted Latex project proposal form, discussion of FS-code
20/10/2021	limited progress of the code side due to ill health, talked about GeNN and code on Github
29/10/2021	Talking about the interim report, walked through GeNN code that Jamie had used on the GitHub, outlined aims to go though the GeNN tutorials
05/11/2021	Talking about the interim report, limited progress of the code side due to ill health
11/11/2021	Submission of the Interim Report
18/11/2021	Discussion about the development of Floating-Point Neuron which will be compatible with GeNN, No Loops
02/12/2021	Need to focus on learning GeNN, discussion about testing dataset distribution to hypothesis how they may impact the approaches performance
09/12/2021	Development of Input Neuron for FP
27/01/2022	Discussion on how synapse / weight update model is used within GeNN, begin development for FP neuron
03/02/2022	Another discussion on how synapse / weight update model is used within GeNN, troubleshoot code
17/02/2022	Focusing on development of FP neuron connected with a single other neuron. Start development of Inter-FP neuron
03/03/2022	Continued development of FP inter - neuron, discussion to alternative methods for this approach
17/03/2022	Paused development of FP neuron. To gain a greater understanding of GeNN, working on a small network of FS neurons was decided.
31/03/2022	Working on a small network of FS neurons, single neuron firing, now adding to a network by connecting with synapses.
08/04/2022	The death of a close friend, an aunt, a grandmother and another aunt within a few weeks resulted in a period of exceptional circumstances, where progress on this project as well as other tasks were put on pause
30/06/2022	Resumed work, focused on getting FS working on MNIST and CIFAR10. Completed FS small network consisting of 3 neurons with 2 synapses connected linearly. Development of FP Input neuron and report started

References

[1] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search", *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. DOI: 10.1038/nature16961.

[2] J. Mattheij, *Jacques mattheij*, https://jacquesmattheij.com/another-way-of-looking-at-lee-sedol-vs-alphago/. (visited on 09/11/2021).

[3] J. Ling, *Power of a human brain*, https://hypertextbook.com/facts/2001/JacquelineLing.shtml, 2001. (visited on 09/11/2021).

[4] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design", *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018. DOI: 10.1109/ijcnn.2018.8489241.

[5] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures", *Frontiers in Neuroscience*, vol. 13, 2019. DOI: 10.3389/fnins.2019.00095.

[6] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes", *Nature Machine Intelligence*, vol. 3, no. 3, pp. 230–238, 2021. DOI: 10.1038/s42256-021-00311-4.

[7] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]", *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. DOI: 10.1109/MSP.2012.2211477.

[8] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)", [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html (visited on 09/11/2021).

[9] B. Lake, R. Salakhutdinov, and J. Tenenbaum, "Human-level concept learning through probabilistic program induction", English (US), *Science*, vol. 350, no. 6266, pp. 1332–1338, Dec. 2015, ISSN: 0036-8075. DOI: 10.1126/science.aab3050.

[10] *Bcs code of conduct*. [Online]. Available: https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/ (visited on 09/11/2021).

[11] Genn-Team, *Genn/license.txt at master · genn-team/genn*, Mar. 2017. [Online]. Available: https://github.com/genn-team/genn/blob/master/LICENSE.txt (visited on 09/11/2021).

[12] Sharmaroshan, *Mnist-dataset/license at master · sharmaroshan/mnist-dataset*. [Online]. Available: https://github.com/sharmaroshan/MNIST-Dataset/blob/master/LICENSE (visited on 09/11/2021).

[13] Wichtounet, *Cifar-10/license at master · wichtounet/cifar-10*. [Online]. Available: https://github.com/wichtounet/cifar-10/blob/master/LICENSE (visited on 09/11/2021).

[14] Brendenlake, *Omniglot/license at master · brendenlake/omniglot*. [Online]. Available: https://github.com/brendenlake/omniglot/blob/master/LICENSE (visited on 09/11/2021).

[15] C. Wilson, G. Serrano, A. Koulakov, and D. Rinberg, "A primacy code for odor identity", *Nature Communications*, vol. 8, Nov. 2017. DOI: 10.1038/s41467-017-01432-4.

[16] *Alphago: The story so far*. [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far (visited on 09/11/2021).

[17] F. Paredes-Valles, K. Y. W. Scheper, and G. C. H. E. D. Croon, "Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 2051–2064, 2020. DOI: 10.1109/tpami.2019.2903179.

[18] E. Yavuz, J. Turner, and T. Nowotny, "Genn: A code generation framework for accelerated brain simulations", *Scientific Reports*, vol. 6, no. 1, 2016. DOI: 10.1038/srep18854.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[20] C. D. Wilson, G. O. Serrano, A. A. Koulakov, and D. Rinberg, "A primacy code for odor identity", *Nature Communications*, vol. 8, no. 1, p. 1477, Dec. 2017, ISSN: 2041-1723. DOI: 10.1038/s41467-017-01432-4.

[21] *Omniglot about*, https://omniglot.com/about.htm. (visited on 09/11/2021).

[22] J. C. Knight, A. Komissarov, and T. Nowotny, "Pygenn: A python library for gpu-enhanced neural networks", *Frontiers in Neuroinformatics*, vol. 15, p. 10, 2021, ISSN: 1662-5196. DOI: 10.3389/fninf.2021.659005. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2021.659005.

[23] 2022. [Online]. Available: https://github.com/genn-team/ml_genn/blob/master/ml_genn/layers/fs_input_neurons.py.

[24] 2022. [Online]. Available: https://github.com/genn-team/ml_genn/blob/master/ml_genn/layers/fs_neurons.py.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv 1409.1556*, Sep. 2014.

[26] T. C. Wunderlich and C. Pehle, "Event-based backpropagation can compute exact gradients for spiking neural networks", *Scientific Reports*, vol. 11, no. 1, 2021. DOI: 10.1038/s41598-021-91786-z.

[27] G. Bellec, F. Scherr, A. Subramoney, *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons", *Nature Communications*, vol. 11, no. 1, 2020. DOI: 10.1038/s41467-020-17236-y.