Advancements in Image Classification: Comparing Spiking, Convolutional, and Artificial Neural Networks

Jinchen Luo

Received May 13, 2024
Accepted July 24, 2024
Electronic access August 15, 2024

Neural networks are currently widely used in image classification. Two major network structures, fully connected artificial neural networks (ANNs) and convolutional neural networks (CNNs), complete this task. Another type of neural network, spiking neural networks (SSNs), can accomplish the same task. SNNs process and transmit information via spikes, which mimic brain behaviors. So far, SNNs have been found in various applications, especially in neuromorphic computing, and have demonstrated a power efficiency advantage on neuromorphic chips. This paper compares SNNs with ANNs and CNNs on commercial hardware rather than neuromorphic hardware to evaluate SNNs' capabilities for image classification. We compared the networks' performance on static and neuromorphic event-driven datasets through three different experiments. Our experiments suggested that compared to traditional forms of neural networks namely ANN and CNN on commercial hardware, SNNs have reached comparable accuracy. However, the advantage of its power efficiency on neuromorphic hardware is not shown on commercial hardware with image classification tasks. We have seen an SNN consuming 142% more power, and 128% more memory in training with longer training times compared to a CNN in image classification. We believe the efficiency of SNNs on image classification needs further assessment on neuromorphic hardware in the future.

Introduction

Neural networks have been instrumental in solving various computer vision problems, such as image classification. Image classification is a process where a computer system is trained to recognize and categorize images into predefined classes or labels. It involves feeding a neural network with a large dataset of labeled images, allowing it to learn patterns and features that define each category. Advancements in deep learning, particularly convolutional neural networks (CNNs), have significantly improved image classification accuracy. In 2014, Russakovsky, Olga et al[1] first tested the human annotation accuracy on the ImageNet 2012[2] dataset. It was estimated that a trained human expert can achieve a top-5 error rate of 5.1% in classifying the images. In 2015, this was first surpassed by He, Kaiming, et al[3] using the new activation function PreLU. They achieved a 4.94% top-5 error rate on the ImageNet dataset, marking a significant milestone in image classification. Image classification utilizing neural networks has found applications in many fields. One of the applications is real-time object detection. Redmon et al. (2016) introduced 'You Only Look Once' (YOLO)[4], a unified, real-time object detection system. In the traditional object detection approach, the object's region in an image is first identified using the sliding window technique that moves a window across the image at various scales and aspect ratios to detect objects. Then, a neural network extracts features from the identified region. This approach is computationally expensive since many windows need to be processed to determine a region where objects exist. Furthermore, the traditional approach neglects global contexts and only focuses on local features within proposed regions. This can lead to misclassification, especially in cluttered scenes. YOLO applies a single neural network to an entire image, dividing it into regions and predicting bounding boxes and probabilities for each region' everything happens simultaneously, making it computationally efficient. The efficiency allows YOLO to process images in real-time at a rate of 45 frames per second, significantly faster than its predecessors. Furthermore, the predictions are informed by the global context of the picture, resulting in a higher Mean Average Precision (MAP) compared to its predecessors.

Image classification is also widely applied to the medical field. Li et al. (2020) developed a fully automatic framework to detect COVID-19 using chest CT scans[5]. The authors utilized a modified ResNet-50[6] as the deep learning model. The extracted features from CT scans from ResNet are then fed to a fully connected layer and softmax function to classify each type (COVID-19, community-acquired pneumonia, and non-pneumonia). The deep learning model can accurately detect COVID-19 and differentiate it from community-acquired pneumonia and other lung diseases, helping for early diagnosis of the disease, which is important for treatment and the isolation of the patients to prevent the virus from spreading. Furthermore, image classification is applied in astrophysics. Zhu, Xiao-Pan, et al[7]. discuss using deep CNNs for galaxy morphology classi-

fication[8], focusing on classifying galaxies into five categories based on their shapes. The dataset used in the study is drawn from Galaxy Zoo 2, containing 61578 galaxy images with morphological classifications. The network used in the study is a variant of ResNets V2, which is a deep residual network architecture explicitly designed for galaxy classification, different from the conventional ResNet. This network achieved higher accuracy than un-modified network structures, such as ResNet or VGG, making it more applicable for large-scale galaxy classification in forthcoming surveys such as the Large Synoptic Survey Telescope (LSST).

Widely used datasets to train neural networks for image classification are MNIST[9] and CIFAR-10[10]. MNIST consists of 70,000 handwritten digit images (60,000 for training, 10,000 for testing), each representing a digit from 0 to 9. The images are grayscale, 28x28 pixels in size, and have been preprocessed to be centered and normalized. Figure 1A shows an example image from the MNIST dataset with label 5. LeNet-5[9] was one of the first CNNs to be trained on the MNIST dataset. Its pioneering architecture contributed to the development of deep learning, especially in the field of image recognition.

On the other hand, the CIFAR-10 dataset consists of 60,000 color images, each 32x32 pixels in size, divided into 10 different classes, such as airplanes, trucks, birds, and dogs. The dataset is split into 50,000 training images and 10,000 test images. Figure 1B shows an example of the CIFAR-10 dataset. Compared to MNIST, this dataset is considered to be more challenging as the images are more complex because of its bigger image size, multiple color channels, object complexity, and variable backgrounds. Even though many famous image classification networks like LeNet, AlexNet[11], and ResNet are built on the concept of CNNs, another type of network can complete such tasks, namely Spiking Neural Networks (SNNs). The concept of an event-driven neural network known as the SNN has been proposed to mimic brain behavior. In SNNs, neurons are designed to imitate the behavior of brain neurons and use spikes to transmit and process information. SNNs and human brains process temporal information, meaning they can handle data that varies over time. Operating via spikes, which can be coded as 1s and 0s rather than floats (decimals) and theoretically requires a smaller amount of computation power and memory to train SNNs. SNNs are also compatible with specialized neuromorphic event-driven hardware. Event-driven hardware is designed to process and respond to discrete events or input signals rather than operating on a fixed clock. This allows for the development of low-power, real-time, and scalable neural computing systems. Furthermore, SNNs are more biologically plausible than traditional ANNs, as they incorporate the fundamental principles of how biological neurons process and transmit information. This can lead to a better understanding of the brain's functions as it replicates the brain's neural dynamics and potentially inspires new approaches to artificial intelligence.

So far, SNNs have found their place in many applications like audio signal processing[12] and real-time object detection[13,8,14]. In traditional image classification, the input to the model is typically a static image, represented as a grid of pixel values. In contrast, SNNs can work with neuromorphic datasets; they use a different representation, known as event-based or spike-based data. In this representation, each pixel in the image is associated with a series of events, where an event represents a change in the pixel's intensity over time. An example of a widely used neuromorphic dataset is the N-MNIST dataset. N-MNIST[15] or Neuromorphic MNIST, is a variant of the classic MNIST. It consists of 70,000 sequences of events, each obtained by performing calculations on the original MNIST dataset. There are 60,000 sequences of events for training and 10,000 for testing. NMNIST dataset that is designed to simulate the output of neuromorphic sensors, which mimic the behavior of biological vision systems to train SNNs. Unlike the original MNIST dataset, which consists of static grayscale images, N-MNIST is composed of sequences of events, where each event represents a change in pixel intensity over time. N-MNIST is created by recording the MNIST images with a Dynamic Vision Sensor (DVS) sensor mounted on a motorized pan-tilt unit. This setup mimics human saccadic movements, which are quick, simultaneous movements of both eyes in the same direction. The sensor captures the changes in pixel intensity at a rate of approximately 100 milliseconds over time, resulting in a dataset that represents visual information as temporal sequences of spikes akin to the data processed by biological neurons. This spiking dataset is particularly useful for training SNNs since no conversion is needed; unlike static datasets, pixel intensity has to be converted into spiking intensity and fed into the network recurrently to simulate recurrent static events. Figure 2 is an example of the N-MNIST dataset with events at each saccade.

Training deep SNNs can face various optimization challenges, including vanishing or exploding gradient problems. In deep neural networks, the gradients are propagated backward through the layers during the backpropagation algorithm. In the first case, as the gradients flow through the many layers, they can become increasingly smaller, especially in the lower layers of the network. This can lead to the weights in the lower layers not being updated effectively, causing the network to learn slowly or even stop learning altogether. Exploding gradient is an issue similar to vanishing gradient, only involving a huge gradient, which causes the network to reach an unexpected updated outcome. This problem is significant in SNNs due to the spiking neurons' property of threshold-based activation and the complex dynamics of coding data. A Spike-Element-Wise (SEW) residual network[16] was proposed; it allows for easier identity mapping to address deep SNNs' optimization challenges. This Spike-Element-Wise approach outperformed the previous attempts of Spiking ResNet in terms of accuracy on the ImageNet-2012 dataset, especially in very deep neural networks with more

than 100 layers.

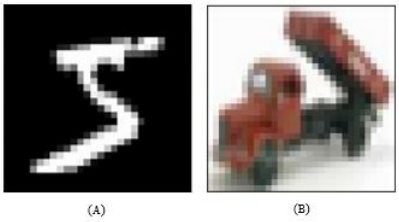

Fig. 1 (A) EXAMPLE OF MNIST DATASET, LABEL = 5. (B) EXAMPLE OF CIFAR-10 DATASET, LABEL - TRUCK

Another optimization challenge SNNs face is the spike generation function used in SNNs. It is non-differentiable and, therefore, not directly compatible with the standard error backpropagation algorithm, which uses chain rule to optimize parameters. Shrestha, Sumit B., and Garrick Orchard introduced the Spike Layer Error Reassignment (SLAYER)[17] in 2018, a novel method for error backpropagation in SNNs. It distributes error credit through the layers of SNN, similar to how traditional backpropagation works. However, unlike backpropagation in conventional neural networks, SLAYER also distributes error credit back in time because the current state of a spiking neuron depends on its previous states, including the states of its input neurons. SLAYER can learn both synaptic weights and axonal delays simultaneously, which is a unique feature compared to previous works. SLAYER has achieved state-of-the-art accuracy on various benchmark datasets for tasks such as visual digit recognition, action recognition, and spoken digit recognition.

As mentioned. SNNs are capable of achieving real-time object detection. Kim, Seijoon, et al. introduced the Spiking YOLO[13]. The paper proposed two novel methods to make SNN compatible and suitable for object detection: channel-wise normalization and signed neurons with an imbalanced threshold. The channel-wise normalization method ensures that each channel (feature map) carries relevant information efficiently. By normalizing across channels, Spiking-YOLO achieves faster and more accurate communication within the network.

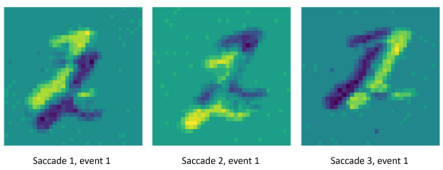

Fig. 2 EVENTS OF EACH SACCADE FROM N-MNIST

Furthermore, Spiking-YOLO leverages signed neurons with imbalanced thresholds. These neurons respond differently to positive and negative inputs, improving the network's discriminative power. The proposed Spiking-YOLO model achieves remarkable results comparable to Tiny YOLO, which is a smaller and shallower version of YOLO with a higher speed for object detection on datasets like PASCAL VOC[18] and MS COCO[19], which are both widely used large datasets for object detection while consuming significantly less energy on neuromorphic chips.

There are a few comparative papers on SNNs and traditional neural networks like ANNs and CNNs; many of them have demonstrated that SNNs running on neuromorphic hardware have an advantage in power efficiency compared to conventional ANNs or CNNs [13,14,20]. This paper compares the performance of SNNs and the performance of ANNs and CNNs in image classification on commercial hardware which are readily available, off-the-shelf computing components marketed for general use, e.g. GPUs.

The accuracy, time of each epoch, power consumption of hardware, and memory consumption during the training, validation, and testing process of three experiments are measured to compare the performance of the networks. The first experiment compares the performance of a simple, fully connected ANN and a fully connected SNN with the same structure. The second experiment compares the performance of a CNN with five convolutional layers and a convolutional SNN with the same structure. Lastly, the third experiment compares the training results of a conventional static dataset and a neuromorphic, event-based dataset.

METHODOLOGY

In this section, we will introduce the implementation of the SNN, how it is trained, and the experiments performed.

SNN IMPLEMENTATION

SNNs operate via electrical impulses known as spikes. To implement spiking neurons, the Leaky Integrate and Fire (LIF)

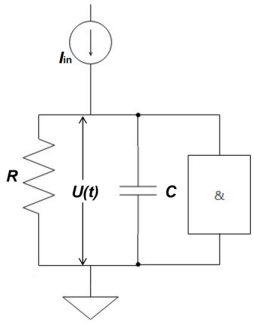

Fig. 3 RC CIRCUIT TO GENERATE SPIKES

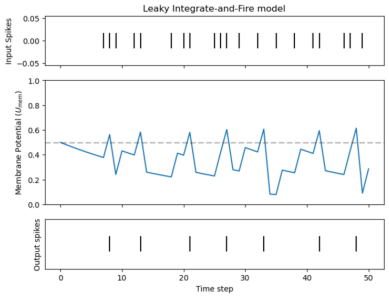

Fig. 4 LEAKY INTEGRATE-AND-FIRE MODEL

model[21] was used for its computational efficiency. Like the ANN's neuron model[22], the spiking neurons use a weighted sum as their input; the main difference is how they process the input. Rather than passing the input through a non-linear, differentiable activation function like a Sigmoid or rectified linear unit (ReLU), the input contributes to a membrane potential U[t].

When the membrane potential reaches or exceeds a threshold θ, the neuron releases a spike to its subsequent connections, contributing to further network actions; Lapicque discovered these dynamics in 1907. He concluded that a spiking neuron coarsely resembles a low-pass filter circuit consisting of a resistor R and a capacitor C, later dubbed the Leaky Integrate-and-fire model[23] (Figure 3). Biologically, the capacitance is from the insulating lipid bilayer forming the membrane of a neuron. The resistance is from gated ion channels, which open and close, modulating the charge carrier's diffusion across the membrane[24]. The dynamics of the passive membrane modeled using an RC circuit can be represented as:

$$\tau \frac{dU(t)}{dt} = -U(t) + I_{\text{in}}(t)R \tag{1}$$

where $\tau = RC$ is the circuit's time constant. The solution of 1 for a constant input is:

$$U(t) = I_{\text{in}}R + [U_0 - I_{\text{in}}R]\, e^{-t/\tau} \tag{2}$$

This demonstrates an exponential relaxation of $U(t)$, and U_0 is the initial membrane potential at $t = 0$. The forward Euler method is used to find an approximate solution to 1 where time is discretized.

$$U[t] = \beta U[t-1] + (1-\beta) I_{\text{in}}[t] \tag{3}$$

In neural networks, weights are typically a learnable parameter. In this case, $(1-\beta)$ is a learnable weight W.

$$I_{\text{in}}[t] = WX[t] \tag{4}$$

This equation removes the effect of β on the input $X[t]$. Accounting for spiking and membrane potential gives the following:

$$U[t] = \beta U[t-1] + WX[t] - S_{\text{out}}[t-1]\theta \tag{5}$$

$$S_{\text{out}}[t] = \begin{cases} 1, & \text{if } U[t] > \theta \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$S_{\text{out}}[t]$ is the output generated by the neuron. If the neuron is activated, the reset term subtracts the threshold from the membrane potential; otherwise, the reset term has no effect. Figure 4 demonstrates that a spiking neuron emits spikes depending on the membrane potential, which is determined by input spikes.

Currently, there are three main ways to encode the input data into spikes: rate coding, latency/temporal coding, and delta modulation[23]. Rate coding converts input intensity into a firing rate of spikes or spike count. Latency coding uses the spike time in a fixed period to code inputs. Delta modulation converts a change in input intensity into spikes. This article will mainly focus on rate coding and will use rate coding for further network training. For rate coding, the number of spikes generated in a unit of time is used to code data. For example, a bright pixel in the MNIST dataset with ten classes of handwritten number images to classify can generate a high-frequency firing rate of spikes. In contrast, a dark pixel will generate a low firing frequency. Rate coding has the advantage in error tolerance and back-propagation. For error tolerance, ideally, if a neuron fails to fire, there are many more spikes to reduce the effect of a misfire. Also, more spikes generated means a stronger gradient signal can be used for learning when back-propagating. Thus resolving the dead neuron problem.

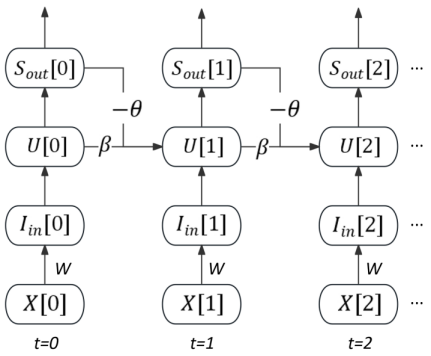

Fig. 5 COMPUTATION GRAPH FOR FORWARD PROPAGATION OF A SNN BASED ON EQUATION (5) AND (6)

In a classification task to decode a rate-encoded output, the neuron that fires with the highest frequency is used as the correct class since each neuron is simulated for the same number of time steps. Two major ways to train an SNN are Shadow training and Backpropagation using spikes. Due to the non-differentiability of spikes, the dead neuron problem arises during error backprop-

agation when updating the weight W in (4):

$$U[t] = \beta U[t-1] + WX[t] - S_{\text{out}}[t-1]\theta \tag{7}$$

This update causes the membrane potential to change; however, this change in membrane potential may not lead to a further change to the spike's presence (6) since the membrane potential may still be inadequate to trigger the release of a spike. This means there is no adequate learning signal when backpropagating through the network, leading to a dead neuron problem. Shadow training is one approach to address the dead neuron problem. It uses a traditional ANN to train and convert it into an SNN[20,25]. One advantage of shadow training is that the results from traditional ANN can be converted and utilized efficiently in the inference of SNN. However, this technique does not use the SNN's advantage of temporal dynamics. Also, long conversion steps could offset the power-efficient benefits of using an SNN.

Another method is back-propagation using spikes. This is the most commonly adopted method for recent SNNs. Figure 5 demonstrates a computation graph illustrating the forward propagation of each time step of a neuron in SNN. In back-propagation, the gradient flows from the loss to all descendants; thus, the chain rule can be applied iteratively. The present-time application of W is referred to as the immediate influence, with the historical application of W described as the prior influence.

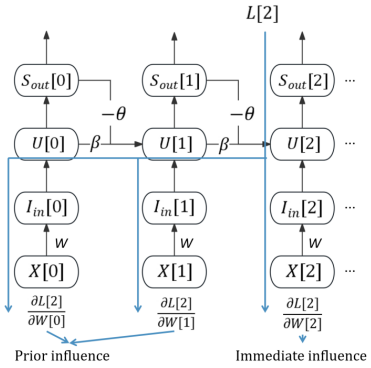

Fig. 6 COMPUTATION GRAPH FOR BACKWARD PROPAGATION OF AN SNN

Figure 6 presents various pathways by which gradients $\frac{\partial L}{\partial W}$ can be passed, allowing the use of gradient descent to train a network. The parameter W is applied at each time step, denoted as $W[s]$, and a loss $L[t]$ is calculated. Thus, depending on the calculated loss, each weight application will only affect the present and future losses. The influence of $W[s]$ on $L[t]$ when $s = t$ is the immediate influence, whereas when $s < t$ is the prior influence. The influence of all parameter applications on present and future losses is summed together to define the global gradient:

$$\frac{\partial L}{\partial W} = \sum_t \left(\frac{\partial L[t]}{\partial W} \right) = \sum_t \sum_{s \le t} \left(\frac{\partial L[t]}{\partial W[s]} \frac{\partial W[s]}{\partial W} \right) \tag{8}$$

The weights will also be shared among all time steps, $W[1] = W[2] = \cdots = W$, thus:

$$\frac{\partial W[s]}{\partial W} = 1 \tag{9}$$

which simplifies (8) to:

$$\frac{\partial L}{\partial W} = \sum_t \sum_{s \le t} \left(\frac{\partial L[t]}{\partial W[s]} \right) \tag{10}$$

The act of thresholding the membrane potential is functionally equivalent to applying a step function: If a membrane potential exceeds the threshold, the neuron releases a spike; if not, no action is taken. This function is nondifferentiable. To address this, a surrogate gradient approach is proposed. During the forward propagation, the unit step function is still applied using the membrane potential and threshold to determine spikes from a neuron. However, the step function is substituted during the backward propagation with continuous, differentiable functions, e.g., Sigmoid or Arctan. The derivative of the continuous function is used as a substitute. Arctan will be used as the default surrogate gradient function for the experiments conducted in the paper:

$$s = \frac{1}{\pi} \tan^{-1}(\pi U) \quad (11) \tag{11}$$

The derivative of s is:

$$\frac{\partial s}{\partial U} = \frac{1}{\pi(1 + (\pi U)^2)} \quad (12) \tag{12}$$

which can be substituted as:

$$\frac{\partial S}{\partial U} \leftarrow \frac{\partial s}{\partial U} \quad (13) \tag{13}$$

A major advantage of using the surrogate gradient approach is that it helps to overcome the dead neuron problem by enabling errors to propagate to earlier layers, regardless of spiking.

EXPERIMENTS

We compare the performance of SNN against ANNs and CNNs on both traditional and neuromorphic datasets through three different experiments. We run all experiments on a system with Windows 10 with a single Nvidia RTX 3060 TI with 8 GB of video memory, 32GB of RAM, and an 8-core AMD 5800x CPU using Pytorch[26] and SNNtorch[23]; Weights & Biases is used to record data and create plots. During the training process, the following data are recorded: network accuracy and time consumption for each epoch while training and testing; GPU video

memory usage; and GPU power consumption. For training, the weights are initialized using the Kaiming initializing technique[3]. Cross-entropy loss is used as the function for the fully connected ANN and the CNN. For the three SNNs used, the decay rate for the spiking neurons is set to 0.8, and MSE count loss[17] is the loss function. The correct class has a target firing rate of 80% of all time steps, and incorrect classes are set to 20%. This setting encourages incorrect neurons to fire at a lower frequency rather than not spiking at all, avoiding dead neurons caused by a minimized weight, leading to a non-spiking neuron.

In total, three datasets are used to train, validate, and test the performance of the networks: MNIST, CIFAR-10, and N-MNIST. To train all three networks, 80% of the training set of the datasets will be separated randomly for training, and the rest 20% will be used for validating the networks after each epoch of training. During validations, there are no updates in the weights. Then the parameters are saved to allow for testing on the test set. To compare the neural networks, we also compute the recall and F1 score at inference time. The recall is calculated by:

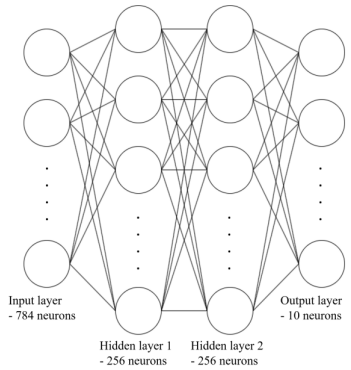

Fig. 7 STRUCTURE OF FULLY CONNECTED ANN FOR EXPERIMENT 1. CONSISTS OF 784 INPUT NEURONS TO HOLD INPUT FROM THE MNIST DATASET, TWO HIDDEN LAYERS WITH 256 NEURONS EACH, AND AN OUTPUT LAYER WITH TEN NEURONS REPRESENTING TEN CLASSES TO CLASSIFY

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad (14) \qquad (14)$$

And the F1 score is calculated as follows:

$$F_1 = \frac{2 \times \text{accuracy} \times \text{recall}}{\text{accuracy} + \text{recall}} \quad (15) \qquad (15)$$

Experiment 1

We test a fully connected ANN with the structure of 784-256-256-10: an input layer with 784 input neurons, followed by two hidden layers with 256 neurons, and an output layer with 10 neurons representing the 10 classes to be classified (Figure 7). This was compared against a fully connected SNN with the same structure on the MNIST dataset. Since the dataset is already in a grayscale form (pixel intensity ranging from 0 to 1), there are no transformations to the dataset and each pixel's intensity is translated into spikes and directly fed into the network. Both networks are trained for 20 epochs using the Adam optimizer[27],. ReLU[28] is used as an activation function for the fully connected ANN. The learning rate is set to 0.001.

Experiment 2

We test a CNN with five convolutional layers and two fully connected layers (Figure 8). All kernels have a size of 5×5 for extracting detailed features, a stride of 1 and a padding of 2 is also used. A max-pooling layer with a kernel size of 2×2 and a stride of 2 is used after every convolutional layer to amplify the features. The first convolutional layer has 3 input channels representing the training dataset's three color channels: R, G, and B. Each pixel's intensity from the color channels is converted into spikes and fed into the network. The last convolutional layer is then connected to a fully connected layer with 512 neurons. To reduce overfitting, this layer also serves as a dropout layer at a rate of 0.2, which means that 20% of the weights in the layer will be set to 0 to reduce overfitting. The 512 neurons are then connected to the output layer, with ten neurons representing ten classes to be classified. The same structure was used on an SNN. The networks are trained on the CIFAR-10 dataset. Both networks are trained for 30 epochs using the Adam optimizer. ReLU is used as an activation function for the CNN. The learning rate is set to 0.001.

Experiment 3

We compare the training effect of neuromorphic datasets with traditional static datasets on SNNs. We use a CNN and an SNN with two convolutional layers and one fully connected layer (Figure 9). All kernels have a size of 5×5, stride of 1, and no padding. A max-pooling layer with a kernel size of 2×2 and a stride of 2 is used after every convolutional layer. The last convolutional layer is then connected to a fully connected layer with 800 neurons. We use MNIST as the static dataset and N-MNIST as the neuromorphic dataset. Each pixel's intensity is converted into spikes and fed into the network. Before feeding the N-MNIST dataset into the network, the dataset is first converted into a denser representation to reduce computation requirements by integrating the events happening within 1000 s into one frame. Then, the events will be denoised. Denoise removes isolated, one-off events. If no event occurs within a neighborhood of 1 pixel across 10000 s, the event is filtered by setting the pixel's intensity to zero. The MNIST dataset is enlarged to a resolution of 32×32 to match the resolution of N-MNIST. The network is trained for 10 epochs using the Adam

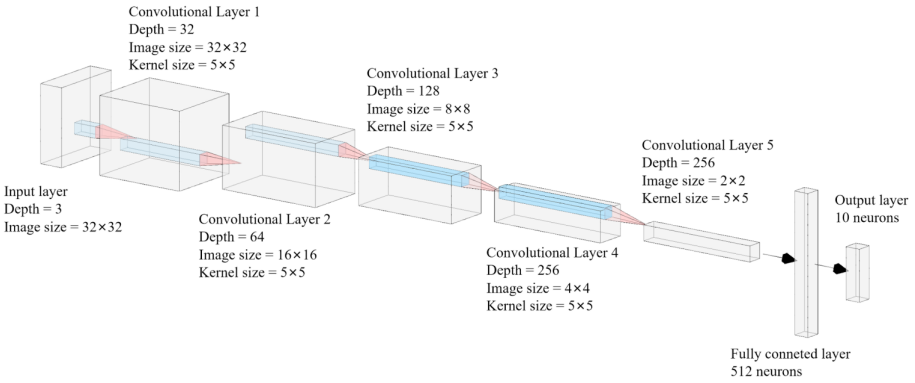

Fig. 8 STRUCTURE OF CNN FOR EXPERIMENT 2. CONSISTS OF FIVE CONVOLUTIONAL LAYERS WITH A KERNEL SIZE OF 5*5, THEN THE FINAL CONVOLUTIONAL LAYER IS CONNECTED TO TWO FULLY CONNECTED LAYERS FOR CLASSIFICATION.

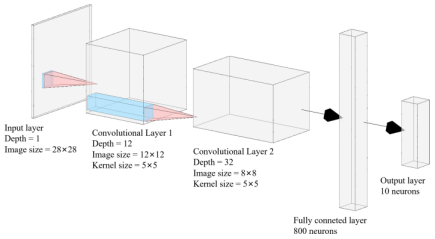

Fig. 9 STRUCTURE OF CNN FOR EXPERIMENT 3. CONSISTING OF TWO CONVOLUTIONAL LAYERS WITH A KERNEL SIZE OF 5*5, THE SECOND CONVOLUTIONAL LAYER IS CONNECTED TO TWO FULLY LAYERS FOR FINAL CLASSIFICATION

optimizer. The learning rate is set to 0.001.

RESULTS

Experiment 1

Figure 10A reports the fully connected ANN's accuracy and the SNN's accuracy on the training and validation set. The ANN reached 99.86% on the testing set and 97.84% accuracy on the validation set. The network used 20 epochs to reach optimum accuracy on the validation set. The SNN reached 99.93% accuracy on the testing set and 98.25% accuracy on the validation set. The network used 13 epochs to reach optimum accuracy on the validation set. Figure 10B reports the time elapsed for each epoch of training both networks. On average, the ANN used 9.39s for each epoch and the SNN 26.38s. The ANN consumed a maximum of 9.53s at the 2nd epoch, making a 1.4% difference compared to the average and the SNN consumed 27.10s at the 1st epoch, making a 2.6% difference compared to the average. Figure 10C reports the GPU memory allocated while training both networks. Over the 20 epochs, training the ANN consumed 13.6% of video memory, and the SNN consumed 13.6%. Figure 10D reports the power usage while training and testing both networks. On average, over the 20 epochs, the ANN uses 16.2W and the SNN 21.1W (Table I).

The ANN reached 97.81% accuracy on the test set and used 1.33s to enumerate through the dataset. Meanwhile consuming 41.8W of power and 12.9% of video memory. It got a recall score of 0.978 and a F1 score of 0.977. The SNN reached 98.05% accuracy on the same dataset and used 3.16s to enumerate through the dataset. Meanwhile consuming 44.4W of power and 13.3% of video memory. It got a recall score of 0.980 and a F1 score of 0.981 (Table II).

Experiment 2

Figure 11A reports the accuracy of CNN and SNN on the training and validation set. The CNN reached 98.36% accuracy on the testing set and 70.05% accuracy on the validation set. The network used 7 epochs to reach its optimum validation accuracy. The SNN reached 86.19% accuracy on the testing set and 67.73% on the validation set. The network used 30 epochs to reach its optimum validation accuracy. Figure 11B reports the time elapsed for each epoch of training for both networks. The CNN, on average, took 6.42s for each epoch, and the maximum time consumption of 10.61s occurred at the 1st epoch, accounting for a 39% difference compared to the average. The SNN, on average, took 81.15s for each epoch, and the maximum time consumption of 81.43s occurred at the 1st epoch, accounting for a 0.3% difference. Figure 11C reports the GPU memory allocated while training and testing both networks. Over the 30 epochs, training the ANN consumed 16.2% of video memory, and training the SNN consumed 39.3%. Figure 11D reports the power usage while training and validating both networks. On average, over the 20 epochs, the CNN used 58.5W, and the SNN used 133.2W(Table I).

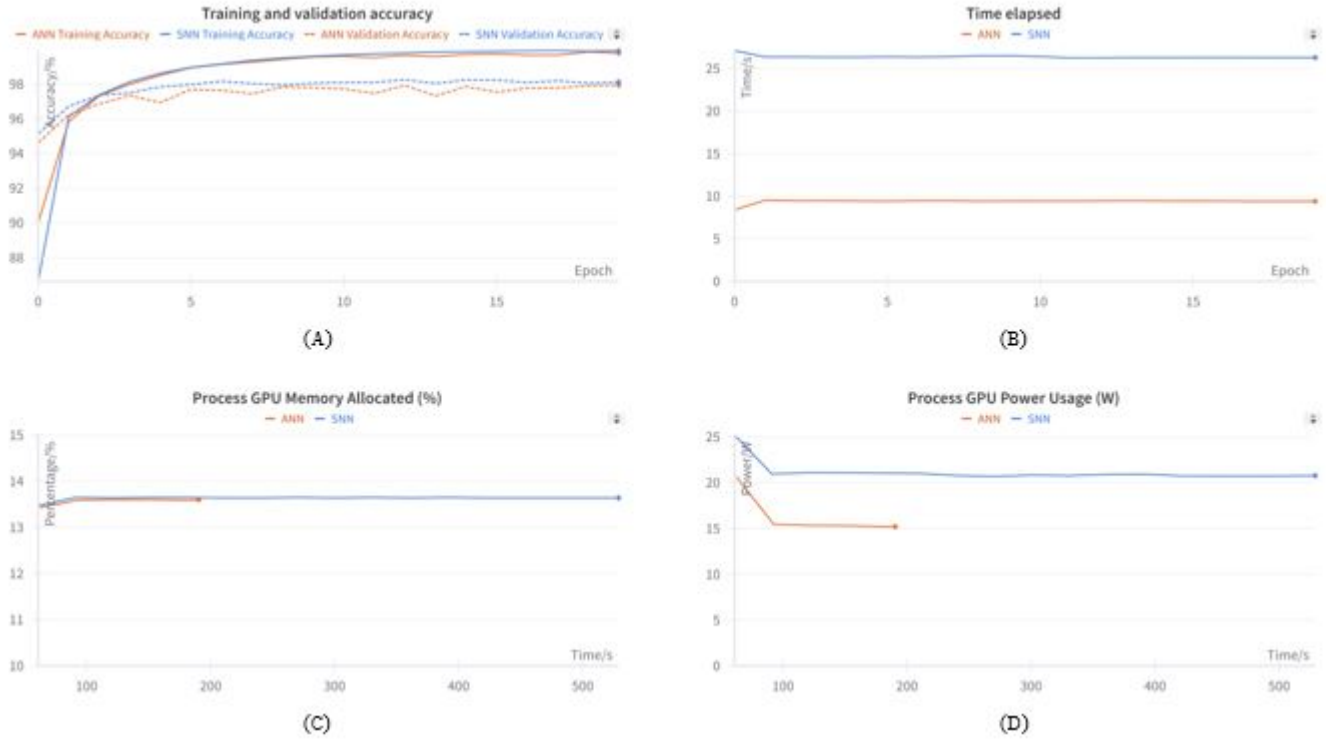

Fig. 10 RESULTS FOR EXPERIMENT 1. (A) VALIDATION ACCURACY OF EACH EPOCH. (B) TIME ELAPSED OF EACH EPOCH (TRAINING AND VALIDATION). (C) GPU MEMORY ALLOCATED FOR TRAINING AND VALIDATION. (D) GPU POWER USAGE FOR TRAINING AND VALIDATION. ORANGE LINES: FULLY CONNECTED ANN. BLUE LINES: FULLY CONNECTED SNN.

The CNN reached 69.74

Experiment 3

Figure 12A reports the accuracy of the CNN trained on MNIST and N-MNIST on the training and validation set Running with the MNIST dataset, the CNN reached 98.72% accuracy in testing and 98.45% accuracy in validation. The network used 9 epochs to reach its optimum validation accuracy. The CNN running with the N-MNIST dataset reached 97.33% accuracy in training and 97.44% accuracy in validation. The network used 10 epochs to reach its optimum validation accuracy. Figure 12B reports the time elapsed for each epoch of training and validation for the CNN running with both datasets. On average, the CNN running with MNIST used 285.87s to train, and the CNN running with N-MNIST used 583.76s. The CNN running with MNIST consumed a maximum of 289.58s at the 1st epoch and the SNN consumed 1188.27s at the 1st epoch. Figure 12C reports the GPU memory allocated while training with both datasets. On average, over the 10 epochs, training the CNN with MNIST consumed 50.0% of video memory, while the CNN with N-MNIST consumed 82.4%. Figure 12D reports the power usage while training and testing CNN running with both datasets. On average, over the 10 epochs, the CNN with MNIST used 80.2W, and the CNN with N-MNIST used 72.7W(Table I). On the test set, the SNN trained on MNIST reached 98.61% accuracy and used 29.04s to enumerate through the dataset. Meanwhile consuming 46.7W of power and 15.8% of video memory. It got a recall score 0.986 and a F1 score of 0.986 on the test set. The SNN reached 97.52% accuracy on the same dataset and used 64.24s to enumerate through the dataset. Meanwhile consuming 44.2W of power and 29.0% of video memory. It got a recall score of 0.975 and a F1 score of 0.975 on the test set (Table II).

DISCUSSION AND CONCLUSION

In this paper, we compared the performance of SNNs, ANNs, and CNNs in image classification on commercial hardware through three experiments.

It can be concluded from the first experiment that the SNN has obtained a similar accuracy on the validation set compared to the fully connected ANN. Though the ANN have demonstrated a slightly more significant overfitting problem than the SNN as the 2.02% difference in accuracy between training and validation is more than the 1.68% difference from the SNN. The

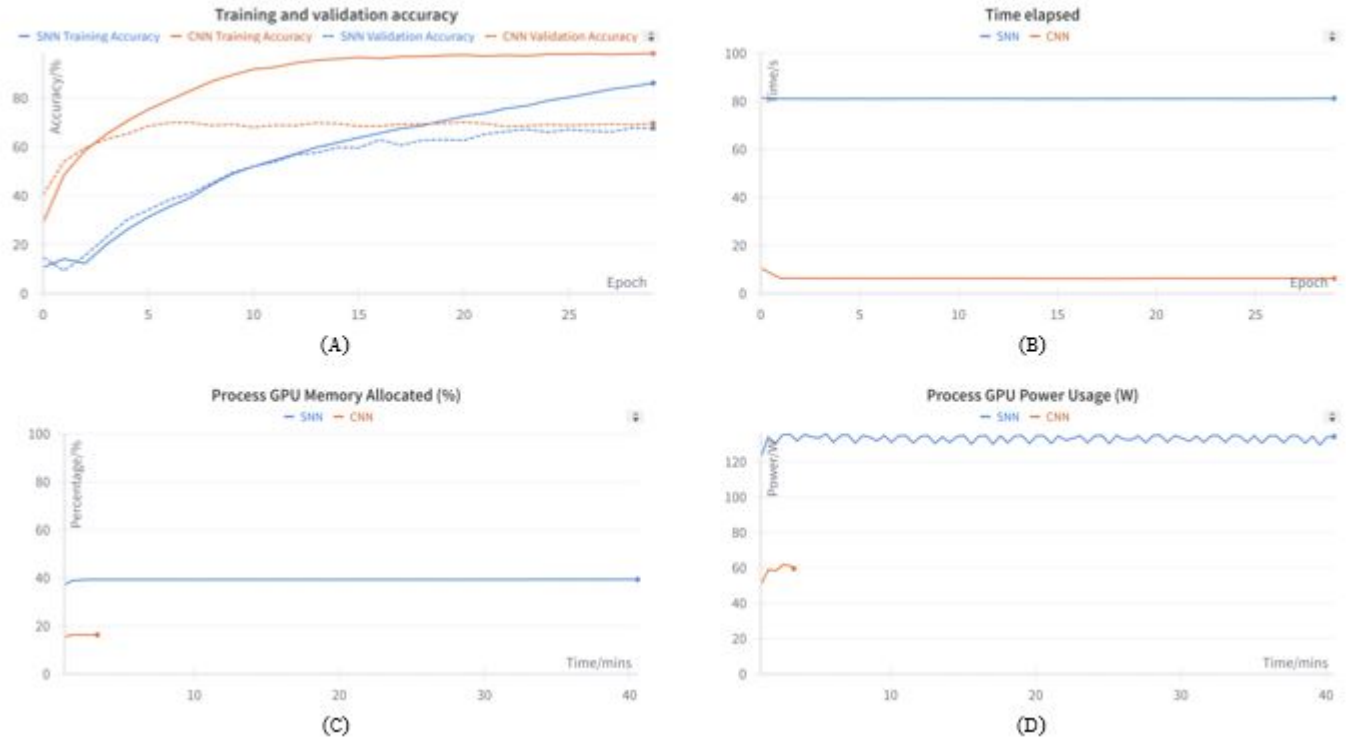

Fig. 11 RESULTS FOR EXPERIMENT 2. (A) VALIDATION ACCURACY OF EACH EPOCH. (B) TIME ELAPSED OF EACH EPOCH (TRAINING AND VALIDATION). (C) GPU MEMORY ALLOCATED FOR TRAINING AND VALIDATION. (D) GPU POWER USAGE FOR TRAINING AND VALIDATION. ORANGE LINES: CNN. BLUE LINES: SNN

SNN consumed the same amount of GPU video memory during training and validation (Table I). However, the SNN took 181% more time to train and validate each epoch. Also, the SNN consumed 30.2% more power during training than the ANN under the same conditions (Table I). In terms of time variability on the training set, both networks have shown a consistent time consumption across the epochs. On the test set, the SNN also obtained a similar accuracy compared to the ANN while consuming a similar amount of memory, power, and more time. For the ANN and SNN based on the information from the recall and the F1 score, both networks performed well on the test data with balanced results. The networks are well-balanced, neither of which overly cautious (high accuracy, low recall) nor overly eager to make positive predictions (low accuracy high recall) (Table II).

The second experiment shows a similar pattern. On the training and validation set, the SNN reached a similar accuracy compared to the CNN. However, the SNN took more epochs to reach optimum accuracy. Though both networks are showing overfitting at some significance with the CNN being more significant than the SNN. The CNN has an accuracy difference of 28.31% in between training and validation, more than the 18.46% from the SNN. In terms of time variability, the CNN's time consumption across different epochs vary more significantly than the SNN. Furthermore, the SNN used more than 10 times the amount time compared to the CNN, 142% more power, and 128% more memory (Table I). On the test set, the SNN also obtained a similar accuracy compared to the CNN while consuming a similar amount of memory, but with more power and more time. Based on their recall and F1 score, both networks are performing moderately well on the test set. With the CNN performing slightly better at distinguishing false positives and negatives (Table II).

In the first experiment, the SNN reached the optimal accuracy in fewer epochs compared to the ANN. The temporal nature of a SNN makes them capture temporal dynamics better than ANN which leads to a faster convergence speed. It is also necessary to point out that the SNN's actual time consumption per epoch is higher compared to the ANN, making the total time consumption greater.

For both experiments, we believe that the power inefficiency and the large time consumption come from training SNNs on

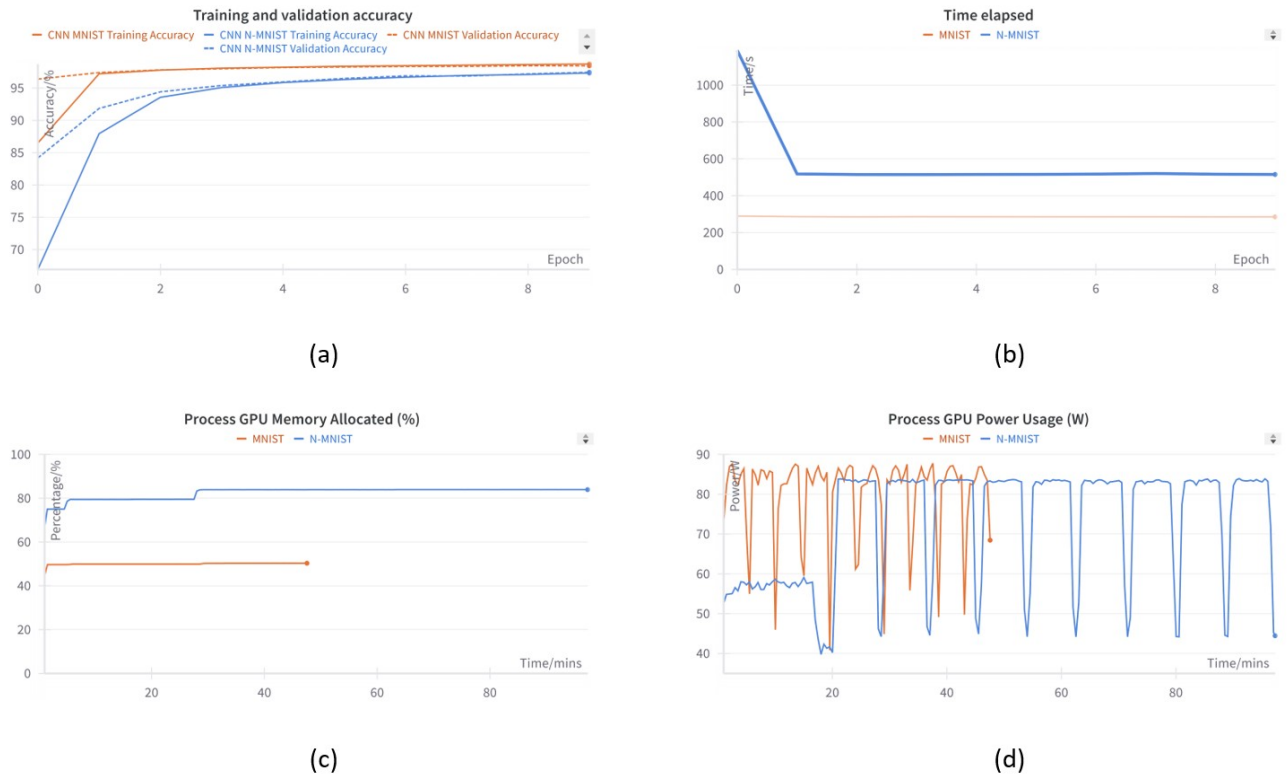

Fig. 12 RESULTS FOR EXPERIMENT 3. (A) VALIDATION ACCURACY FOR EACH EPOCH. (B) TIME ELAPSED FOR EACH EPOCH (TRAINING AND VALIDATION) (C) GPU MEMORY ALLOCATED FOR TRAINING AND VALIDATION. (D) GPU POWER USAGE FOR TRAINING AND VALIDATION. ORANGE LINES: CNN WITH MNIST. BLUE LINES: CNN WITH N-MNIST

commercial hardware rather than neuromorphic hardware. First, unlike traditional artificial neurons, which compute an output value once each time they receive input, spiking neurons need to integrate inputs over time, maintaining and updating their membrane potentials continuously with every timestep until a spike is triggered. This continuous updating process requires repetitive computations that consume computational resources and power. Furthermore, each neuron's membrane potential acts as a state variable that must be stored and accessed at each timestep. This repeated memory access for a potentially large number of neurons can be inefficient. Furthermore, during backpropagation, the surrogate gradient approach is used. Calculating the surrogate gradients adds computational complexity due to the need to handle both the actual spiking dynamics and the surrogate gradients during training. This double handling increases the computational workload and hence the energy consumption.

Lastly, instead of calculating a single weight at each neuron in conventional neural networks, the back-propagation of SNN needs to calculate multiple weights at different time steps and add them together to form a single weight update, which requires applying multiple chain rules and, thus, requires more computation. The third experiment demonstrated that using a neuromorphic dataset like N-MNIST can still reach a similar optimum accuracy, recall score and F1 score with a similar number of epochs to converge and a smaller power consumption than converting static datasets like MNIST into spikes. However, this approach requires significantly longer time and larger memory to train. We believe this phenomenon comes from the dynamic nature of N-MNIST. For a single piece of training data, only one grayscale image is needed for MNIST, while loading the N-MNIST data involves loading a series of different spike events. This means loading the full dataset into memory takes longer, slowing down the training process. We can also see a relatively lower GPU power consumption in the first epoch and a longer time elapsed (Figure 12D). We believe this phenomenon is because the program loads the dataset while training. Once the full dataset is loaded, the training process returns normal. To conclude, although SNNs can reach similar training results compared to conventional neural networks, their power efficiency on commercial software is still questionable. The efficiency of running SNNs on neuromorphic chips like TrueNorth[29] is not brought to commercial hardware. Further tests on neuromorphic hardware are needed to assess the full capabilities of SNNs. To address the efficiency challenges of

Table 1 Results obtained from training the six networks in three experiments on the training and validation set

Experiment	Network	Highest training accuracy/%	Highest validation accuracy/%	epochs to reach optimum validation accuracy	Average time consumption per epoch/s	Average memory consumption/%	Average power consumption/W
1	ANN	99.86	97.84	20	9.39	13.6	16.2
	SNN	99.93	98.25	13	26.38	13.6	21.1
2	CNN	98.36	70.05	7	6.42	16.2	58.5
	SNN	86.19	67.73	30	81.15	39.3	133.2
3	SNN: MNIST	98.72	98.45	9	285.87	50	80.2
	SNN: N-MNIST	97.33	97.44	10	583.76	82.4	72.7

Table 2 Results obtained from testing the six networks in three experiments on the testing set

Experiment	Network	Testing accuracy/%	Time consumption/s	Average memory consumption/%	Average power consumption/W	Testing recall score	Testing F1 score
1	ANN	97.81	1.33	12.9	41.8	0.978	0.977
	SNN	98.05	3.16	13.3	44.4	0.98	0.981
2	CNN	69.74	1.11	16.8	52.3	0.698	0.696
	SNN	67.89	7.93	17.4	114.4	0.679	0.677
3	SNN: MNIST	98.61	29.04	15.8	46.7	0.986	0.986
	SNN: N-MNIST	97.52	64.24	29	44.2	0.975	0.9765

running SNNs on commercial hardware, future research should focus on the development and testing of SNNs on dedicated neuromorphic chips. This could potentially lead to significant improvements in power efficiency and computational speed. Additionally, investigating alternative learning algorithms that are more suited to the temporal dynamics of SNNs could further enhance their performance. It would also be beneficial to expand the scope of datasets used in training and testing to include more complex and varied image sets, which could provide a more comprehensive understanding of the capabilities and limitations of SNNs.

Acknowledgements

Thank you for the guidance of Chiara Di Vece, mentor from University College London, in the development of this research paper.

References

1 O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, 2015, https://arxiv.org/abs/1409.0575.

2 J. Deng, R. Socher, L. Fei-Fei, W. Dong, K. Li and L.-J. Li, 2009 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2009, pp. 248–255.

3 K. He, X. Zhang, S. Ren and J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015, https://arxiv.org/abs/1502.01852.

4 J. Redmon, S. Divvala, R. Girshick and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016, https://arxiv.org/abs/1506.02640.

5 L. Li, L. Qin, Z. Xu, Y. Yin, X. Wang, B. Kong, J. Bai, Y. Lu, Z. Fang, Q. Song, K. Cao, D. Liu, G. Wang, Q. Xu, X. Fang, S. Zhang, J. Xia and J. Xia, *Radiology*, 2020, **296**, 200905.

6 K. He, X. Zhang, S. Ren and J. Sun, author, 2016, pp. 770–778.

7 X.-P. Zhu, J.-M. Dai, C.-J. Bian, Y. Chen, S. Chen and C. Hu, , 2019, **364**, 55.

8 S. Zhou, Y. Chen, X. Li and A. Sanyal, *IEEE Access*, 2020, **8**, 76903–76912.

9 Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, *Proceedings of the IEEE*, 1998, **86**, 2278 – 2324.

10 A. Krizhevsky, *University of Toronto*, 2012.

11 A. Krizhevsky, I. Sutskever and G. E. Hinton, Advances in Neural Information Processing Systems, 2012.

12 J. Timcheck, S. B. Shrestha, D. B. D. Rubin, A. Kupryjanow, G. Orchard, L. Pindor, T. Shea and M. Davies, *Neuromorphic Computing and Engineering*, 2023, **3**, 034005.

13 S. Kim, S. Park, B. Na and S. Yoon, *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, **34**, 11270–11277.

14 A. Kugele, T. Pfeil, M. Pfeiffer and E. Chicca, in *Hybrid SNN-ANN: Energy-Efficient Classification and Object Detection for Event-Based Vision*, Springer International Publishing, 2021, p. 297–312.

15 G. Orchard, A. Jayawant, G. K. Cohen and N. Thakor, *Frontiers in Neuroscience*, 2015, **9**, year.

16 W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li and Y. Tian, *Science Advances*, 2023, **9**, eadi1480.

17 S. B. Shrestha and G. Orchard, *SLAYER: Spike Layer Error Reassignment in Time*, 2018, https://arxiv.org/abs/1810.08646.

18 M. Everingham, L. Van Gool, C. Williams, J. Winn and A. Zisserman, *International Journal of Computer Vision*, 2010, **88**, 303–338.

19 T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, *Microsoft COCO: Common Objects in Context*, 2015, https://arxiv.org/abs/1405.0312.

20 P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni and E. Neftci, *Conversion of Artificial Recurrent Neural Networks to Spiking Neural Networks for Low-power Neuromorphic Hardware*, 2016, https://arxiv.org/abs/1601.04187.

21 A. Burkitt, *Biological cybernetics*, 2006, **95**, 1–19.

22 S. Agatonovic-Kustrin and R. Beresford, *Journal of Pharmaceutical and Biomedical Analysis*, 2000, **22**, 717–727.

23 J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong and W. D. Lu, *Proceedings of the IEEE*, 2023, **111**, 1016–1054.

24 A. Hodgkin and A. Huxley, *Journal of Physiology*, 1952, **117**, 500–544.

25 Y. Hu, H. Tang and G. Pan, *IEEE Transactions on Neural Networks and Learning Systems*, 2023, **34**, 5200–5205.

26 A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019, https://arxiv.org/abs/1912.01703.

27 D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017, https://arxiv.org/abs/1412.6980.

28 V. Nair and G. E. Hinton, Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.

29 F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson and D. S. Modha, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, **34**, 1537–1557.