Static hand gesture recognition for American sign language using neuromorphic hardware

Mohammadreza Mohammadi1, Peyton Chandarana1,∗, James Seekings2, Sara Hendrix and Ramtin Zand2

University of South Carolina, Columbia, SC 29201, United States of America

∗ Author to whom any correspondence should be addressed.

1 These authors contributed equally.

E-mail: psc@email.sc.edu

and ramtin@cse.sc.edu

Keywords: sign language recognition, neural compute stick, neuromorphic hardware, spiking neural network, assistive technology, Intel Loihi

Abstract

In this paper, we develop four spiking neural network (SNN) models for two static American sign language (ASL) hand gesture classification tasks, i.e., the ASL alphabet and ASL digits. The SNN models are deployed on Intel’s neuromorphic platform, Loihi, and then compared against equivalent deep neural network (DNN) models deployed on an edge computing device, the Intel neural compute stick 2 (NCS2). We perform a comprehensive comparison between the two systems in terms of accuracy, latency, power consumption, and energy. The best DNN model achieves an accuracy of 99.93% on the ASL alphabet dataset, whereas the best performing SNN model has an accuracy of 99.30%. For the ASL-digits dataset, the best DNN model achieves an accuracy of 99.76% accuracy while the SNN achieves 99.03%. Moreover, our obtained experimental results show that the Loihi neuromorphic hardware implementations achieve up to 20.64× and 4.10× reduction in power consumption and energy, respectively, when compared to NCS2.

1. Introduction

Sign language is a visual language that enables people who are deaf or hard of hearing to communicate with others in their communities. To convey emotion, grammar, and sentence structure similar to spoken language, sign language employs visual and manual elements such as hand gestures, facial expressions, and body movements. Hand gestures are regarded as the fundamental component of a sign language vocabulary. In addition to these gestures, facial expressions and body movements are used to accentuate the emotions of words and phrases [1]. Hand gestures in a sign language can be classified as static or dynamic depending on whether or not hand motion is incorporated into the sign interpretations [2, 3]. Static hand gestures, also known as static hand postures, consist of the shape and orientation of the hand and fingers and are commonly used for fingerspelling alphabet letters and digits. Dynamic hand gestures, on the other hand, are a set of hand gestures accompanied by motion for word interpretation and translation.

Automatic hand gesture recognition is applicable to a variety of applications and has been a broad and significant research area over the last decade [4]. The most widely researched approaches to automatic hand gesture recognition employ motion sensors and vision sensors.

Motion sensor techniques collect and track hand gestures, finger orientation, and hand velocity using a range of sensors such as sensor gloves [5], motion controllers [6], accelerometers [7], magnetometers, and gyroscopes [8, 9]. Many works in the past have adopted multiple motion sensors simultaneously to increase the classification accuracy of the performed signs [10–12]. Employing multiple motion sensors in a recognition system allows for a more precise recognition in particular sign language vocabularies but consequentially results in higher implementation cost and complexity. While motion sensor-based approaches have the benefit of being able to accurately capture motion data in real-time, these sensors are typically expensive and inconvenient for signers since installing or wearing these instruments can restrict motion and thus hinder performance.
On the other hand, vision-based approaches utilize at least one camera and image processing algorithms including segmentation, shape detection, motion tracking, color detection, and contour modeling to capture and recognize actual hand gestures without the need for cumbersome physical devices [13–16]. Since these approaches do not require wearing the devices to track motion, they are more user-friendly and comfortable without impeding the wearer’s motions. Similar to many other object detection and recognition applications, deep learning methods, particularly convolutional neural networks (CNNs), have recently become the primary choice for sign language applications [17–20]. However, the major downside of deep learning-based techniques is that they often consume a significant amount of power [21]. Since hand gesture recognition models are typically intended to be deployed on portable devices like smartphones, tablets, and wearable devices like smart glasses, power and energy are major constraints. Low-power machine learning is an important field of study that can enable such applications at the edge devices. In particular, neuromorphic algorithms and hardware, which are inspired by the low-power processing of the human brain, have been attracting more attention in recent years [22, 23].

In 2018, Intel introduced a neuromorphic processor called Loihi capable of running spiking neural networks (SNNs) with orders of magnitude lower power consumption while achieving accuracies close to state-of-the-art CPU, GPU, and TPU implementations [24]. Along with Loihi, Intel provided a python library, NxSDK, for SNN design and deployment. NxSDK included many features for designing SNNs from scratch including setting learning rules for synaptic plasticity learning methods. This design methodology, however, does introduce its own complexities in design, training, and hyper-parameter tuning to get accurate and meaningful results.

While there are inherently many differences between how artificial neural networks (ANNs) and SNNs are trained and operate, conversion processes have been widely researched to convert ANNs to SNNs to reduce the complexity of the end-to-end design of SNNs. In 2017, Rueckauer et al [25] developed the SNN conversion toolbox which aimed to convert trained ANN models to SNNs. While the concept of converting ANNs to SNNs was not novel at the time with many works introducing different conversion methods [26, 27], the SNN conversion toolbox intended to streamline the process for conversion by both creating a more automatic conversion pipeline and also implementing several features missing in the previous conversion works. Unlike previous conversion methods, the SNN conversion toolbox allows an ANN to be trained in a deep learning library like TensorFlow [28] or PyTorch [29] with layers commonly found in CNN architectures. The toolbox implements spiking layers like average pooling and convolutional layers to provide a means for the models to be parsed and converted to SNNs without as much hyperparameter tuning and as many design considerations. In 2021, Rueckauer et al [30] introduced NxTF which extended the SNN conversion toolbox to enable the deployment of converted SNNs to Intel’s Loihi neuromorphic chip. NxTF was designed specifically to map layers found in the ANN models to layers that Loihi could understand through a custom NxSDK backend [23].

In this paper, we focus on classifying static images of the ASL alphabet and ASL digits. The following are our contributions:

• We design four ANN models and train each model with the ASL alphabet and ASL digit static image datasets for a total of eight models. We then convert them to SNNs using the SNN conversion toolbox.
• We analyze the trade-off between accuracy and latency and compare the differences when favoring accuracy or latency.
• We investigate methods for accurately measuring power and energy consumption.
• We compare and analyze the hardware performance of the ANN models on the Intel neural compute stick 2 (NCS2) to the SNN models on Intel Loihi in terms of accuracy, latency, power consumption, and energy.

The subsequent sections of this paper are organized as follows. The datasets for this study are presented in section 2. Section 3 describes the ANN model design. Section 4 provides a summarized background on the SNN conversion toolbox’s conversion methodology. Section 5 introduces the experimental methodology performed in this work including the processes for hardware deployment and measuring latency, power, and energy. The results of the experiments are discussed and evaluated in section 6. Section 7 concludes the paper and provides some discussion for future study.

2. ASL datasets

In this paper, two different static image datasets are studied. The American sign language (ASL) alphabet [31] is the first dataset and replaces the common handwritten digit dataset, MNIST [33], commonly used as an ANN model’s proof-of-concept dataset in classification. The ASL alphabet dataset includes static images of multiple people repeating ASL finger-spelling against various backdrops. With the exception of J and Z, which require
motion, the ASL alphabet dataset of hand gestures is a multi-class problem with 24 classes of letters. Figure 1(a) depicts examples of each class. The ASL alphabet dataset, created by [31], was constructed by considerably expanding a small set of 1704 color images that were not cropped around the hand region of interest. The designers of the dataset, [31], generate additional data using an image processing pipeline which is comprised of cropping the images around the hands, converting the images to gray-scale, resizing the images, and then making more than 50 variations to increase the dataset size. Furthermore, the data augmentation includes the use of Mitchell, Hermite, and Catrom filters, coupled with 5% random noise, ±15% brightness/contrast, and three degrees of image rotation. After these data augmentation steps, the dataset includes 34,627 gray-scale images of size 28 × 28 pixels which are split into the separate train, validation, and test datasets. 27,455 samples, ∼80%, of the ASL alphabet are used for training and validation, and 7,172 samples, ∼20%, are used for testing. Figure 1(b) shows the distribution of training samples across 24 classes.

The other dataset we employ in this study is the ASL digits dataset [32]. It is comprised of 2,062 RGB images with 100 × 100 pixels that are divided into ten classes, digits 0–9. We resized the resolution of these images to 28 × 28 and converted the RGB images to gray-scale. 20% of this dataset is used as the test dataset (413 images) and the remaining images are used for validation, 330 images, and training. Figure 1(c) shows a sample image for each class.

3. Model design

We use three CNN models and a multi-layer perceptron (MLP) neural network for the static ASL image classification task. The CNN models are inspired by three standard and well-known models, i.e., LeNet, AlexNet, and VGGNet. Compared to the original models, we slightly modified our models to create smaller versions that still achieve high accuracy values. Moreover, we constrained the models such that they could be readily converted to SNN models. The specifics for each implementation are provided in the following subsections.

3.1. MLP

Figure 2(a) shows the structure of the MLP that we used in this work. The proposed MLP contains two hidden layers with 512 and 256 neurons each, as well as 24 or ten output neurons for the ASL alphabet or ASL digits dataset, respectively. The first and second layers are followed by a dropout layer each with a probability of 0.2.
3.2. LeNet
The architecture of the Lenet [34] model that we employed here is shown in Figure 2(b). It is comprised of two convolutional layers and three fully connected (FC) layers. The first and second convolution layers consist of six and 16 kernels of size $5 \times 5$, respectively. After the first and second convolution layers, there is a non-overlapping average pooling layer with a $2 \times 2$ filter size and strides of 2. The three FC layers each include 120, 84, and 24 or ten neurons depending on which dataset is used. In this study, we modified the original LeNet model and applied two dropout layers with a probability of 0.25 to prevent overfitting after the first and the second FC layers.

3.3. AlexNet
The AlexNet model [35] consists of five convolutional layers each followed by max pooling and three FC layers, all of which utilize the ReLU activation function except the output layer, which uses the softmax activation function. The original AlexNet model is quite large, with over 62 million parameters. Here, we changed the structure of the original network as follows. Instead of 96 $11 \times 11$ kernels with a stride of 4, just 6 $5 \times 5$ kernels with a stride of 1 are used for the first convolution layer. Instead of 256 kernels, we employ 12 $5 \times 5$ kernels for the second convolution layer. We utilize 24 kernels with a size of $3 \times 3$ for each of the next three convolution layers. In the three FC layers, we employ 120, 84, and 24 or ten neurons. To avoid overfitting, we apply dropout to the FC layers with a probability of 0.5 similar to the original AlexNet. Figure 2(c) shows the AlexNet-inspired architecture.

3.4. VGGNet
Another well-known CNN used herein is VGGNet [36], which employs several $3 \times 3$ filters instead of larger filters like AlexNet [35]. VGGNet, like AlexNet, uses ReLU activation functions in the network’s hidden layers. Depending on the number of layers, there are multiple variations of VGG architecture. VGG-16, for example, includes 16 layers and about 138 million parameters. Figure 2(d) depicts the structure of the VGG-inspired model developed in this work. We utilize only nine layers consisting of six convolution layers, and three FC layers. We use six kernels for the first layer, 16 kernels for the second layer, 32 kernels for the third and fourth layers, and 48 kernels for the fifth and sixth layers. A non-overlapping max pooling layer with a $2 \times 2$ filter...
size and strides of 2 follow each convolution block. For the FC layers, we use the same number of neurons as AlexNet and LeNet models.

3.5. Model training
The proposed models are trained using TensorFlow 2.6.2 [37] with a categorical cross-entropy loss function, a batch size of 128, and the Adam optimizer with a learning rate of 0.001 on the ASL alphabet and ASL digits datasets for 50 and 400 epochs, respectively. The intensities of the input images in both datasets are normalized from 0–255 to 0–1. Furthermore, data augmentation is utilized, which involves randomly rotating images in the range of ten degrees, randomly shifting images horizontally and vertically by 10%, and randomly zooming images by 10%. During training, the best epoch is saved for each of the proposed models based on the lowest validation loss. Metrics like accuracy, precision, recall, and F1-score are used to evaluate the models.

4. SNN conversion toolbox background
The SNN conversion toolbox proposed in [25] was designed with flexibility in mind to enable ANN models to be automatically converted to SNNs [25]. During the SNN conversion process the toolbox performs several different steps:
(a) Parse the trained ANN model along with the trained weights and activation values.
(b) Normalize, scale, and set the spiking neuron parameters such as the membrane potential thresholds, biases, and weights.
(c) Convert the layers of the ANN model into equivalent spiking representations using methods such as convolution unrolling.
(d) Deploy the resulting SNN on neuromorphic hardware or use an SNN simulator.

The SNN conversion toolbox employs rate-based encoding which generates input spike trains with regular spiking frequencies over a duration parameter set in toolbox configuration settings. According to the SNN conversion toolbox documentation [25] this duration value correlates to the number of timesteps that each input is exposed to the SNN, in milliseconds, during inference. While the duration parameter is noted to be measured in milliseconds, this time is not to be confused with the network’s real-world latency. As we will see later in section 6.2 the input duration and the latency are correlated in a linear fashion, but they are not equal in a 1-to-1 sense. The duration parameter does, however, affect the length of an input spike train to be proportional to the timesteps specified. These spike trains then propagate through the network causing neurons in the SNN to fire if the neuron membrane potential is driven to or above its threshold. Then in the final layer, the classification layer, the neuron which fires the most is taken as the output class. These steps are a high-level view of the SNN conversion toolbox’s ANN conversion methodology. We refer the reader to [25] for detailed information on the methodologies used during the conversion process.

5. Experiment methodology
Now that neuromorphic hardware, such as Intel’s Loihi [24], is becoming more readily available to researchers, SNNs can now be designed and tested on hardware specifically designed to accelerate SNNs. This allows SNNs to be fairly compared against their deep neural network (DNN) counterparts on conventional hardware accelerators like GPUs and TPUs [38, 39].

In this work, we deploy our developed ANN and SNN models on the Intel NCS2 and the Intel Loihi neuromorphic platform, respectively. Our experimental methodology is depicted in figure 3. In particular, the Loihi experiments were performed on four Loihi chips for the accuracy measurements and a board of 32 Loihi chips called Nahuku-32 for power measurements. Each of the models in figure 2 were deployed on Loihi and NCS2 using the SNN conversion toolbox and OpenVINO APIs, respectively. From there, the performance of each hardware platform was measured with respect to latency, power, and accuracy.

5.1. Intel NCS2
The Intel NCS2 is based on the Intel Movidius X vision processing unit, which has 16 programmable SHAVE cores and a dedicated neural compute engine for hardware acceleration of DNN inferences. It features a base frequency of 700 MHz with a 16 nm technology node. Additionally, NCS2 has 4 GB memory with a maximum frequency of 1600 MHz. NCS2 supports 16 bit floating point operations, which we have found to be sufficient for our ASL classification tasks [40].
Figure 3. Experimental methodology for ASL classification on Intel NCS2 and Loihi. ANN models are first created based on their deployment platform. For Loihi, we constrain the networks by replacing $\tanh$ and $\text{MaxPooling}$ with $\text{ReLU}$ and $\text{AveragePooling}$, respectively, and then train the networks. Both sets of models are then converted to using respective tools for their platform. For Loihi, we run the model multiple times with varying durations and optimize the duration parameter by minimizing the duration and maximizing accuracy. Finally, the models are deployed on their respective platforms.

| Node      | Chips | x86 Cores | Neurocores | Neurons | In-axons | Out-axons | State memory (KB) |
|-----------|-------|-----------|------------|---------|----------|-----------|------------------|
| Loihi     | 4     | 12        | 512        | 524,288 | 2,097,152| 2,097,152 | 65,536           |
| Nahuku-32 | 32    | 96        | 4,096      | 4,194,304| 16,777,216| 16,777,216| 524,288          |

5.1.1. Conversion
To deploy the ANN models on NCS2, Intel provides a Python3 and C library called OpenVINO. Specifically, we used OpenVINO 2022.1.4 for our experiments. To deploy the models to the NCS2, the TensorFlow trained models were first frozen using TensorFlow’s `get_concrete_function()` and `convert_variables_to_constants_v2()` functions. From these frozen models, the OpenVINO model optimizer was used to convert the models to a format that enables their deployment on the NCS2. Once the model is optimized for Intel NCS2, we then used the inference engine API built into OpenVINO to perform inference and subsequently measure the accuracy, latency, and power.

5.1.2. Latency and power measurement
Using OpenVINO, we performed ten iterations over the entire test datasets, 7172 images for the ASL alphabet dataset and 413 images for the ASL digit dataset. The latencies were recorded over the ten iterations for each image and then averaged at the end. We used a USB 3.0 measurement tool (MakerHawk UM34C [41]) along with a phone application to measure and record the power of the NCS2. This measurement device captures the overall system power usage, including USB I/O, due to its inline nature. The power measurement tool was attached to a computer’s USB port along with the NCS2 and the idle and running power was recorded. The idle power was recorded for 5 min after plugging the NCS2 into the power measurement tool and the computer, and the voltage and current are measured every second. We then perform inference on the entire test dataset for ten iterations and record the average running power. The difference between average running power and idle power is calculated as the inference power.

5.2. Intel Loihi
Each experiment was performed on Intel’s Neuromorphic Research Community’s (INRC) cloud infrastructure for INRC members to test models on Intel’s Loihi platform. In particular, two different nodes in the INRC cloud were used, one consisting of four Loihi chips and the other, Nahuku-32, consisting of 32 Loihi chips combined on a single board [23]. Each Loihi neuromorphic core or neuro-core can simulate up to 1024 spiking neurons or compartments, 4096 fan-out axons, 4096 fan-in axons, and 128 kilobytes of fan-in state memory [24]. The details of each Loihi platform node in the INRC cloud infrastructure are provided in Table 1.

5.2.1. Conversion methodology
In 2021, NxTF [30] was released enabling the SNN conversion toolbox [25] to convert a trained ANN model to an SNN and allow it to be deployed on Intel’s Loihi platform. The SNN conversion toolbox, while supporting many of the common layers found in CNNs, does not support some ANN components and layers on Loihi. To combat this, we employ a constrain-then-train method to ensure compatibility with the Loihi backend for the SNN conversion toolbox before conversion. The constrain-then-train method consists of first replacing
Table 2. Loihi neuron parameters.

| Parameter                          | Value | Description                                                                 |
|-----------------------------------|-------|-----------------------------------------------------------------------------|
| Reset modea                       | Soft  | Resets membrane potential to $u(t) = u(t-1) - v_{Th}$                       |
| Threshold-input ratioa            | 1     | Balances $v_{Th}$ w.r.t. input voltage change                               |
| Compart. bias exponenta           | 6     | $B_{bias} = \langle biasMant \rangle^{2} \langle biasExp \rangle^{2}$       |
| Voltage threshold mantissaa       | 512   | Used to compute voltage threshold $v_{Th} = \langle v_{ThMant} \rangle^{2} v_{Th}$ |
| Weight bit precisiona             | 8     | Bit precision to represent weight connections                              |
| Bias bit precisiona               | 12    | Bit precision to represent bias connections                                 |
| Compart. voltage decay            | 0     | Voltage decay constant per timestep                                        |
| Compart. current decay            | 4095  | Current decay constant per timestep                                        |
| Refractory delay                  | 1     | Enables refractory period after compartment spiking                        |
| Functional state                  | 2     | Idle compartment behavior waiting for excitatory stimulus                   |

*a*Configurable parameters through the SNN conversion toolbox.

Table 3. ASL alphabet model parameters and Loihi core partitioning.

| Layer   | MLP Param | Loihi cores | LeNet Param | Loihi cores | AlexNet Param | Loihi cores | VGGNet Param | Loihi cores |
|---------|-----------|-------------|-------------|-------------|---------------|-------------|--------------|-------------|
| Conv1   | —         | —           | 156         | 9           | 156           | 12          | 60           | 12          |
| Conv2   | —         | —           | 2416        | 4           | 1812          | 7           | 880          | 7           |
| Conv3   | —         | —           | —           | —           | 2616          | 3           | 4640         | 4           |
| Conv4   | —         | —           | —           | —           | 5208          | 4           | 9248         | 7           |
| Conv5   | —         | —           | —           | —           | 5208          | 4           | 13872        | 2           |
| Conv6   | —         | —           | —           | —           | —             | 3           | 20784        | 3           |
| FC1     | 401920    | 10          | 30849       | 1           | 26040         | 1           | 5880         | 1           |
| FC2     | 131328    | 4           | 10164       | 1           | 10164         | 1           | 10164        | 1           |
| Output  | 6168      | 1           | 2040        | 1           | 2040          | 1           | 2040         | 1           |
| Total   | 539416    | 18          | 45616       | 21          | 53244         | 41          | 67568        | 47          |

Table 4. ASL digits model parameters and Loihi core partitioning.

| Layer   | MLP Param | Loihi cores | LeNet Param | Loihi cores | AlexNet Param | Loihi cores | VGGNet Param | Loihi cores |
|---------|-----------|-------------|-------------|-------------|---------------|-------------|--------------|-------------|
| Output  | 2570      | 1           | 850         | 1           | 850           | 1           | 850          | 1           |
| Total   | 535818    | 18          | 44426       | 21          | 52054         | 41          | 66378        | 47          |

the max-pooling layers with average-pooling layers and then changing the activation functions from $TanH$ to $ReLU$. We then train the models with the same configuration as before and then input the trained models into the SNN conversion toolbox.

Before converting the ANNs into SNNs, the SNN conversion toolbox requires settings and parameters to be initialized in the form of a configuration file. This configuration file configures settings like what simulator to use, the input duration in milliseconds, and the neuron parameters. In most cases, we use the default parameters specified in the SNN conversion toolbox examples. However, particular attention was given to the duration and the Loihi neuron parameters. In \[23, 24, 42\], Loihi neurons are described as a variant of the current-based (CUBA) leaky-integrate-and-fire (LIF) neurons described in \[43, 44\]. The specific Loihi compartment/neuron parameters used in our experiments are provided and described in table 2. For more specific information on CUBA LIF neurons and Loihi, we refer the reader to \[43, 44\] and \[23, 24, 42\].

As mentioned in section 4, after the trained ANN is converted to an SNN, the SNN conversion toolbox uses the SNN parameters and layer information to deploy the models on neuromorphic hardware or a software simulator. In this case, the toolbox uses the information about the converted SNN structure to appropriately partition the Loihi neuro-cores. This partitioning process is described in \[30\] and consists of optimization techniques that ensure that the neuro-cores can communicate efficiently. The specific number of neuro-cores which were partitioned for each layer of our converted SNN models can be seen in tables 3 and 4. Once the partitioning process has succeeded, the SNN conversion toolbox proceeds to deploy the SNN on Loihi and begins performing inferences on the test dataset.

5.2.2. Hardware independent simulation

To gauge the energy efficiency of each network independent of the neuromorphic hardware platform, we simulated the converted SNNs using the INIsim SNN simulator built into the SNN conversion toolbox. In table 5,
Table 5. Average network firing rate.

| Model    | ASL alphabet | ASL digits |
|----------|--------------|------------|
| MLP      | 0.050        | 0.039      |
| LeNet    | 0.358        | 0.313      |
| AlexNet  | 0.098        | 0.182      |
| VGGNet   | 0.062        | 0.039      |

Table 6. Average ASL accuracy & standard deviation.

| Dataset    | Architecture | Accuracy (%) | Standard deviation (%) |
|------------|--------------|--------------|------------------------|
|            | ANN          | C-ANN        | SNN                     | ANN          | C-ANN        | SNN         |
| ASL alphabet | MLP          | 92.28        | 91.18                  | 0.88         | —            | 1.28        |
|            | LeNet        | 98.30        | 97.71                  | 94.82        | 0.43         | 0.91        |
|            | AlexNet      | 98.99        | 99.02                  | 97.50        | 0.68         | 0.53        |
|            | VGGNet       | 99.38        | 99.13                  | 98.82        | 0.31         | 0.42        |
| ASL digits  | MLP          | 86.61        | —                      | 86.05        | 1.06         | —           |
|            | LeNet        | 95.04        | 96.44                  | 95.78        | 0.46         | 1.03        |
|            | AlexNet      | 98.77        | 98.60                  | 97.01        | 0.62         | 0.39        |
|            | VGGNet       | 99.20        | 99.08                  | 97.60        | 0.38         | 0.37        |

we report the average firing rate (AFR) of the network as a whole. This AFR is computed by

\[
AFR = \frac{N_{\text{spikes}}}{\text{Batch} \times M_{\text{neurons}} \times \text{Duration}}
\]

where \(N_{\text{spikes}}\) is the number of spikes generated during the simulation for each inference, \(\text{Batch}\) is the batch size of the input which we set to 1, \(M_{\text{neurons}}\) is the number of neurons in the network, and \(\text{Duration}\) is the simulation duration parameter which we set to 500 ms to obtain an upper bound on the AFR.

5.2.3. Latency and power measurement

After designing the ANNs using the aforementioned constrain-then-train method and converting them to SNNs, we performed multiple experiments with varying durations, a hyper-parameter defined in section 4. The duration point space was then searched to achieve one of two objectives: (1) attain high accuracy without duration considerations or (2) achieve a balance between duration and accuracy. To collect the latency and power during inference, we set the profile_performance configuration setting in the toolbox while targeting the Nahuku-32 board. Nahuku-32 was chosen since it contains the necessary power and latency recording hardware [23]. While the Nahuku-32 board contains 32 Loihi chips in a single server node, according to our experiments all of the models utilize one Loihi chip and less than the 128 neuro-cores available as seen in tables 3 and 4. For each of the varying durations, the latency and power metrics were recorded when the SNN models were deployed on the Nahuku-32 board. After completing all inferences, the toolbox then reports the power usage for the neuro-cores, x86 cores, and the total system power. For each model, power measurements for various durations were averaged together to attain a typical power usage of the model on Loihi regardless of the duration.

6. Results

In this section, we compare the SNNs’ and ANNs’ performance in terms of accuracy, latency, power, and energy consumption. We also provide insights into the techniques utilized to obtain high SNN accuracy and then provide an analysis on the balance between accuracy and latency. Finally, the SNN deployed on Loihi is compared to the ANN deployed on Intel NCS2 in terms of latency, power, and energy.

6.1. Accuracy analysis

SNNs have been shown in previous works to have comparable if not better accuracy than ANNs due to the introduction of noise by the nature of spikes approximating the floating point values in an ANN [39, 45]. Table 6 shows the average accuracy and standard deviation values obtained for the ANN, constrained ANN (C-ANN), and SNN models for the ASL alphabet and ASL digits over ten different trials. The confidence intervals for all models and datasets are calculated in appendix D, which shows a narrow margin of uncertainty.

First, we compare the ANN versus C-ANN model accuracies before conversion. As listed in table 6, the VGGNet ANN models achieve the highest values for accuracy. In particular, for the ASL alphabet, the VGG
Table 7. Previous work model comparison with best accuracies.

| Dataset   | Paper              | Accuracy | Parameters |
|-----------|--------------------|----------|------------|
| ASL-alphabet | Mannan et al [46]    | 99.97    | 2994 649   |
|           | Can et al [47]      | 99.91    | 1335 722   |
|           | Fregoso et al [48]  | 99.98    | —          |
|           | Our ANN             | 99.93    | 67 568     |
|           | Our SNN             | 99.30    | 82150      |
| ASL-digits | Rahman et al [20]   | 99.99    | —          |
|           | Xiao et al [49]     | 99.52    | —          |
|           | Our ANN             | 99.76    | 66 378     |
|           | Our SNN             | 99.03    | 80 960     |

ANN realizes the best average accuracy of 99.38% and standard deviation of 0.31%, while the ASL digit’s VGG ANN has the best accuracy and standard deviation of 99.20% and 0.38% respectively. For both datasets, the ASL alphabet and digits, the MLP networks performed the worst with the lowest accuracies. Since the MLP models did not contain any TanH activations or pooling layers, no constrained models were created, and thus, the C-ANN metrics have been left empty. We can also see that the AlexNet C-ANN on the ASL alphabet performs marginally better when compared to the conventional ANN with a 0.03% difference in accuracy and a 0.15% difference in their standard deviation. Table 6 also shows that the average accuracy of the ASL digits LeNet C-ANN is 1.4% higher than that of its ANN, but the standard deviation of the C-ANN is 0.57% higher than the ANN’s.

In table 6, we also present the average accuracies and standard deviation obtained from the deployed SNN models run on the durations which maximize accuracy. For ASL alphabet, the VGGNet SNN achieves the best SNN average accuracy of 98.82% and standard deviation of 0.56% compared to the other SNN architectures. This also holds true for the ASL digits dataset where the highest SNN average accuracy and standard deviation is 97.60% and 1.21%, respectively. Comparing the VGGNet ANN and SNN models for the ASL alphabet, the SNN loses just 0.56% of its accuracy after conversion. Similarly, the ASL digit VGG SNN loses 1.60% compared to VGG ANN. A similar pattern can be seen for the other models in both datasets where SNN accuracies are lower than the ANNs except the LeNet model on ASL digits. In this case, LeNet outperforms the ANN with a difference of 0.74%, but the C-ANN still surpasses the SNN with a difference of 0.66%. In table 7, we compare our best accuracy results to those of the previous works. We can see that, in comparison to the other works, our ANN and SNN models both realize high accuracy with much smaller networks in terms of network parameters.

In table 7 we compare our accuracy results to previous works. To compare our work with other cutting-edge models, we selected the experiment with the best accuracy out of ten distinct trials. We can see that, in comparison to the other works, the accuracies for both our ANNs and SNNs are rather close to the state-of-the-art models which use the same datasets. Our ASL-alphabet SNN is just 0.68% less accurate than the best ANN and our ASL-digits SNN is only 0.94% less accurate than the best ANN for its dataset. Table 7 also shows that our ANN and SNN models both realize high accuracy with much smaller networks in terms of network parameters.

6.2. Latency

As mentioned in section 4, the duration parameter is the number of timesteps, in milliseconds, that each input is exposed to the SNN. Thus, modifying this duration value should have a linear effect on the inference latency. As mentioned in section 5.2.3, the Nahuku-32 board does contain the power and latency measurement hardware, but the polling is limited to a 30–40 ms time resolution [50]. This limit, thus, prevents the power and latency measurements when running on lower values of duration. To mitigate the polling issue, we performed our experiments using a set of higher and uniformly distributed values of duration to drive the time for inference above the 30–40 ms threshold. For each of the higher durations, the models are run and the latency is recorded and plotted seen as red circles on the graphs in figure 4 (and the figures in appendix A). From there, the least squares method was used to generate the line of best fit. As seen in the graphs of figure 4 (and in appendix A), the relationship between duration and latency is roughly linear for all the models on higher durations. Thus, from here on, we use the fitted line to obtain the SNN inference latency for various duration values.

6.2.1. Loihi accuracy and duration/latency balance

As shown in figure 5, as the duration/latency increases so does the accuracy for various SNN models run on Loihi. However, after a specific duration, the accuracy plateaus and there is little to no gain in accuracy at the
Figure 4. MLP latency vs duration graphs for (a) ASL alphabet and (b) ASL digit dataset. See appendix A for the other models. Legend: ○ test points, ▲ best accuracy, ■ balanced duration–accuracy point.

Figure 5. Accuracy–duration tradeoff. The SNN model’s accuracy on test dataset is related to the number of timesteps in which each sample is given.

Table 8. Relaxed accuracy yielding lower duration.

| Dataset | SNN model | Best accuracy (%) | Relaxed accuracy range durations (ms) |
|---------|-----------|-------------------|--------------------------------------|
|         |           |                   | 0.0% <1.0% <2.0% <3.0% <4.0% <5.0% |
| Alphabet| MLP       | 92.07             | 75 25 20 20 20 20                    |
|         | LeNet     | 94.87             | 130 75 65 55 50 50                   |
|         | AlexNet   | 96.88             | 300 130 115 100 95 90                |
|         | VGGNet    | 99.44             | 250 115 100 95 90 85                 |
| Digits  | MLP       | 86.92             | 45 40 30 20 20 20                   |
|         | LeNet     | 97.34             | 135 120 70 65 55 50                 |
|         | AlexNet   | 98.79             | 110 85 80 75 70 65                  |
|         | VGGNet    | 99.52             | 105 90 80 75 75 70                  |

cost of significantly increasing duration and therefore latency. Thus, we aim to achieve a balance between the accuracy and duration to attain acceptable accuracy while reducing the latency.

The blue triangles ▲ on the graphs in figure 4 (and the figures in appendix A) represent the maximum possible accuracy for the respective models between 5 ms and 300 ms with increments of 5 ms. The quantitative values of the blue point can be found as the ‘best accuracy’ metrics in table 8. While it is possible that a higher accuracy could be achieved after 300 duration timesteps, this accuracy would incur a significantly higher inference latency.
Table 9. ASL alphabet SNN best accuracy metrics versus metrics for balanced accuracy–duration on Loihi.

| SNN model | Duration (ms) | Latency (ms) | Accuracy (%) | Balanced point |
|-----------|--------------|--------------|--------------|---------------|
| MLP       | 75           | 40.06        | 92.07        | 20            |
| LeNet     | 130          | 13.21        | 94.87        | 65            |
| AlexNet   | 300          | 20.69        | 96.88        | 115           |
| VGGNet    | 250          | 14.03        | 99.44        | 100           |

Table 10. ASL digit SNN best accuracy metrics versus metrics for balanced accuracy–duration on Loihi.

| SNN model | Duration (ms) | Latency (ms) | Accuracy (%) | Balanced point |
|-----------|--------------|--------------|--------------|---------------|
| MLP       | 45           | 23.89        | 86.92        | 30            |
| LeNet     | 135          | 13.56        | 97.34        | 70            |
| AlexNet   | 110          | 10.67        | 98.79        | 80            |
| VGGNet    | 105          | 8.72         | 99.32        | 80            |

Table 8 demonstrates the accuracy and duration trade-offs, for multiple relaxed accuracy values ranging from 1.0% to 5.0% accuracy reductions. As seen in table 8 there are significant reductions in duration if a small reduction in accuracy is allowed. In tables 9 and 10, we use the duration point from table 8 where the accuracy drop is no more than 2.0%. This threshold was chosen because it reduces the duration, therefore the latency, without significantly reducing the accuracy. However, another accuracy threshold can be used depending on the application’s sensitivity to accuracy or latency. The green squares on the graphs in figure 4 (and the graphs in appendix A) exhibit the 2.0% relaxed accuracy point. The accuracy, duration, and latency values for this point are provided in tables 9 and 10 as the ‘balanced point’ metrics.

As shown in tables 9 and 10, the balanced point's latency is significantly reduced compared to that of the best accuracy point. The ASL alphabet appears to benefit the most from the accuracy–duration balancing. As seen in table 9, the MLP’s SNN accuracy drops just 1.4% but the duration decreases by 73.33% equating to a 68.44% drop in latency. VGGNet’s accuracy is reduced by 1.74% after balancing, from 99.44% to 97.70%, and the duration is subsequently reduced by 60.0% and latency by 42.77%. The accuracy loss for the other two models, LeNet and AlexNet, is 1.48% and 1.75%, respectively. Their duration reduction is 50.0% and 61.67% equating to a latency reduction of 32.10% and 44.90%, respectively.

For the ASL digits dataset, the MLP SNN has the most significant difference in duration and latency with a 31.81% reduction in latency and a 1.45% drop in accuracy. The duration reductions for LeNet, AlexNet, and VGGNet are 48.15%, 27.27%, and 23.81%, while the accuracy loss is 1.94%, 1.45%, and 1.46%, respectively. The latency reductions for the three SNN models are thus 28.69%, 15.28%, and 10.89%, respectively. Hence, for applications that can tolerate more error, it is evident that lower latency configurations can be employed.

6.3. Intel Loihi and NCS2 compared

Here, we compare the performance of ANNs on Intel’s NCS2 with the performance of SNNs on Intel’s Loihi neuromorphic chips in terms of latency, power, and energy.

To measure the inference power consumption on Loihi, the accuracy–latency balanced SNN models discussed in the previous subsection are evaluated on the Nahuku-32 board. The LeNet SNN, for the ASL alphabet, and MLP SNN, for the ASL digits, have the lowest inference power with 58.06 mW and 60.23 mW, respectively, as seen in tables 13 and 14. To analyze the effect the number of parameters in a network has on the power, latency, and energy we also ran the ASL alphabet LeNet SNN model using varying numbers of kernels in the convolution layers to increase/decrease the total number of parameters without changing the number of layers. As seen in table C1 in appendix C, increasing the number of parameters in a network also increases the power of a network showing that power consumption is not just dependent on model depth. The AlexNet SNN, on the other hand, has the highest inference power, with the ASL alphabet and ASL digits inference powers of 79.09 mW and 90.34 mW, respectively (see tables 11 and 12). For NCS2, an idle power of 635 mW is measured using the method discussed in section 5.1.2. The MLP network has somewhat higher inference power for both datasets compared to all other models running on NCS2, with 878 mW and 849 mW for the ASL alphabet and ASL digits, respectively. On the other hand, VGGNet has the lowest inference power on the ASL alphabet, with an inference power of 825 mW. On ASL digits, however, the CNN architectures have very comparable inference power. In terms of latency, the MLP on the NCS2 has the least latency for both ASL alphabet and ASL digits,
with 2.13 ms and 2.06 ms values, respectively, whereas VGGNet has the greatest latency, with values of 2.59 ms and 2.47 ms, respectively. On both datasets, on Loihi, the VGGNet SNN had the least inference latency. The SNN models running on Loihi, in general, have a greater inference latency, ranging from 8.03 ms to 12.64 ms for the ASL alphabet and 7.77 ms to 16.29 ms for the ASL digits. This higher latency, when compared to NCS2, can be attributed to the rate-based spike coding used by the SNN conversion toolbox as well as the inclusion of three x86 cores in the current implementation of Loihi [23, 39]. In previous works [51], rate-based encoding has been shown to significantly increase latency due to features being encoded into the frequency of spikes in spike trains. These spike trains must then be sufficiently long enough with a sufficient firing rate to propagate through the network to allow the network to classify/learn. This can be seen in figure 6 showing a network’s accuracy being dependent on the input duration i.e. the input spike train length. Moreover, in [39], the authors show that most of the higher latency is attributed to I/O-related tasks such as transferring data to and from synchronous and asynchronous domains as well as between the host CPU and devices. This supports the claims from Intel, in [23], that the current implementation’s x86 cores are sub-optimal and can introduce latency overhead due to the communication differences between synchronous computations on the x86 cores and asynchronous computations on the neuro-cores [23].

In terms of inference energy consumption, the LeNet SNN model on Loihi has the lowest inference energy, with 0.52 mJ and 0.59 mJ for the ASL alphabet and ASL digits, respectively. On the ASL alphabet images, the VGGNet SNN model on Loihi consumes only 0.06 mJ more energy than LeNet but achieves 4.31% higher accuracy. VGGNet also consumes 0.07 mJ more inference energy for the ASL digits, although it is 2.66% more accurate compared to the LeNet SNN. As a result, the VGGNet SNN is the more appropriate choice for applications that require a model with high accuracy and low energy consumption.

A comprehensive comparison of the SNN models implemented on Loihi and ANN models deployed on NCS2 is provided in tables 13 and 14. All of the SNN models on Loihi use less power and energy compared to ANN models. On the ASL alphabet, for example, while VGGNet loses just 1.9% accuracy after converting from the ANN to an SNN, it consumes 11.48 \times \text{less} inference power and 3.69 \times \text{less energy.} For the ASL digits dataset, the VGGNet SNN running on Loihi consumes 9.94 \times \text{less} inference power and 3.15 \times \text{lower} energy compared to VGGNet ANN on NCS2.

| Model        | NCS2 power (mW) | Loihi power (mW) |
|--------------|-----------------|------------------|
|              | Idle | Running | Static | Dynamic | Total | Static | Dynamic | Total |
| MLP          | 635  | 1513    | 0.254   | 18.180  | 18.434 | 15.440 | 27.500  | 42.940 |
| Lenet        | 635  | 1487    | 0.255   | 18.736  | 18.991 | 18.119 | 20.954  | 39.073 |
| AlexNet      | 635  | 1482    | 0.263   | 18.806  | 19.069 | 36.423 | 23.595  | 60.018 |
| VGGNet       | 635  | 1460    | 0.253   | 19.566  | 19.819 | 40.180 | 22.976  | 63.156 |

| Model        | NCS2 power (mW) | Loihi power (mW) |
|--------------|-----------------|------------------|
|              | Idle | Running | Static | Dynamic | Total | Static | Dynamic | Total |
| MLP          | 635  | 1484    | 0.253   | 18.187  | 18.442 | 15.476 | 26.312  | 41.788 |
| Lenet        | 635  | 1473    | 0.262   | 18.835  | 19.097 | 18.602 | 22.831  | 41.433 |
| AlexNet      | 635  | 1472    | 0.258   | 18.848  | 19.106 | 35.699 | 35.535  | 71.234 |
| VGGNet       | 635  | 1475    | 0.267   | 18.964  | 19.231 | 42.423 | 22.890  | 65.313 |

| Benchmarking metric | NCS2 | Loihi | MLP | LeNet | AlexNet | VGGNet | MLP | LeNet | AlexNet | VGGNet |
|---------------------|------|-------|-----|-------|---------|--------|-----|-------|---------|--------|
| Accuracy (%)        | 92.24 | 98.93 | 98.68 | 99.60  | 90.67   | 93.39  | 95.13 | 97.70  |
| Inference power (mW)| 878  | 852   | 857  | 825    | 61.37   | 58.06  | 79.09 | 71.89  |
| Latency (mS)        | 2.13 | 2.50  | 2.59  | 2.59   | 12.64   | 8.97   | 11.40 | 8.03   |
| Inference energy (mJ)| 1.87 | 2.13  | 2.22  | 2.14   | 0.78    | 0.52   | 0.90  | 0.58   |

Table 11. Movidius vs Loihi power comparison on ASL alphabet.

Table 12. Movidius vs Loihi power comparison on ASL digits.

Table 13. Movidious vs Loihi—comprehensive analysis—ASL alphabet.
Table 14. Movidious vs Loihi comprehensive analysis — ASL digits.

| Benchmarking metric       | NCS2 (MLP, LeNet, AlexNet, VGGNet) | Loihi (MLP, LeNet, AlexNet, VGGNet) |
|---------------------------|------------------------------------|-------------------------------------|
| Accuracy (%)              | 86.19, 98.55, 99.03, 99.03         | 85.47, 95.40, 97.34, 98.06          |
| Inference power (mW)      | 849, 838, 837, 840                 | 60.23, 60.53, 90.34, 84.54         |
| Latency (mS)              | 2.06, 2.29, 2.39, 2.47             | 16.29, 9.67, 9.04, 7.77            |
| Inference energy (ml)     | 1.75, 1.92, 2.00, 2.08             | 0.98, 0.59, 0.82, 0.66             |

Table 15. NCS2 vs Loihi performance using edge detected images (ASL alphabet).

| Benchmarking metrics       | NCS2 (MLP, LeNet, AlexNet, VGGNet) | Loihi (MLP, LeNet, AlexNet, VGGNet) |
|---------------------------|------------------------------------|-------------------------------------|
| Accuracy (%)              | 87.72, 87.93, 90.69, 95.18         | 86.44, 64.91, 88.64, 92.78          |
| Inference power (mW)      | 863, 847, 868, 853                 | 42.21, 41.03, 62.93, 61.55         |
| Latency (mS)              | 2.05, 2.35, 2.51, 2.59             | 14.26, 12.84, 19.28, 16.93         |
| Inference energy (ml)     | 1.76, 1.99, 2.18, 2.21             | 0.60, 0.53, 1.21, 1.04             |

6.3.1. Increasing data sparseness

Our experiments thus far have employed ANN models trained on grayscale images of the ASL datasets. However, in some cases, all of the information in these grayscale images may not be necessary to classify the hand gestures. Thus, using similar edge detection and methods to [52], we reduce the information in the images by increasing their sparseness. We then re-trained and converted the models using the edge detected images using a fixed duration of 120 ms and report the metrics in table 15 for the ASL alphabet.

Comparing tables 13 and 15, we see that, in general, using edge detection on the data reduces the power of the SNN by 14.38% to 31.22% compared to the original grayscale images on Loihi. As we can see in table 15, most of the models have higher energy consumption. This is due to the increase in latency resulting from fixing the duration to a constant 120 ms between models. We can also see that the Intel NCS2 does not benefit from the increased sparseness of the images. This results in an even greater difference of 13.79× to 20.64× in power consumption between the Intel NCS2 and Loihi, respectively.

7. Conclusion

In this work, we first trained four different ANN models on two static ASL hand gesture image datasets, the ASL alphabet and ASL digits. We then modified, constrained, and trained versions of four common ANN models to ensure compatibility with the SNN conversion toolbox. We then deployed the converted SNNs on Intel’s neuromorphic processor, Loihi, and benchmarked them against their conventional ANN implementations on the Intel NCS2 edge AI accelerator herein. We then performed an analysis of the correlation between the accuracy and duration/latency of the SNN models and provided a method to find a balanced point according to an application’s tolerance to accuracy or latency. Moreover, we discussed specific mechanisms to accurately measure the power consumption and latency in both neuromorphic and edge computing hardware. Finally, we provided a comprehensive comparison between Loihi and NCS2 in terms of accuracy, latency, power consumption, and energy. In terms of accuracy, the SNN models approach or even surpass ANN models. In terms of latency, Loihi falls behind Intel’s NCS2, however, the power reduction realized by Loihi is so significant (9.26–20.64×) that it could still achieve 1.79–4.10× energy savings compared to NCS2 while executing the ASL classification tasks. Future work includes dynamic hand gesture recognition of ASL using temporal learning rules for SNNs.

Acknowledgments

This work is partially supported by an ASPIRE grant from the Office of the Vice President for Research at the University of South Carolina. Special thanks to the Intel Neuromorphic Research Community (INRC) for providing access to the Loihi chips for the experiments performed in this paper.
Figure A1. ASL alphabet latency vs duration. Legend: • test points, ▲ best accuracy, □ balanced duration–accuracy point.

Data availability statement

The data generated and/or analysed during the current study are not publicly available for legal/ethical reasons but are available from the corresponding author on reasonable request.

Appendix A. Duration vs. latency

See figures A1 and A2

Appendix B. Confusion matrices

To visualize how the ANN and SNN differ on a class-by-class basis we have included difference confusion matrices for each of the models in our experiments. These figures are dubbed ‘difference’ confusion matrices because they represent the difference between two models. When the color value of a pixel is close to zero on the color scale, this conveys that the SNN and ANN or C-ANN were about equal in their performance. The diagonals of these difference matrices are emphasized with white asterisks to show where the SNN and ANN correctly classified the input. If a pixel on the diagonal is bright yellow then the SNN correctly classified more inputs than the ANN or C-ANN. However, if the pixel on the diagonal is dark blue or purple the ANN or C-ANN was more accurate than the SNN at classifying the input. If the pixel does not lie on the diagonal of the matrix and is yellow then this concludes that the SNN was more incorrect than the ANN or C-ANN and vice-versa (see figures B1 and B2).
Appendix C. Parameter size effect on latency and power

To examine the impact of varying parameter sizes on Loihi’s latency and power consumption, we trained three LeNet models. Each of these models have the standard depth of five layers but differ in the number of kernels each of the convolution layers use. According to table C1, increasing the number of parameters increases power consumption and latency. Consequently, the number of network parameters has a direct impact on both latency and power consumption.

Appendix D. Accuracy confidence intervals

For all models, we calculated the confidence intervals [53] with a confidence level of 95%. As shown, the uncertainty margin is smaller for the VGGNet ANN compared to the others. However, as we can see, every model has a narrow margin of uncertainty.

To compute the confidence intervals in figures D1 and D2, we use the following equation [54]:

$$CI = \text{Mean} \pm Z_{score} \times \frac{Sd}{\sqrt{N}}$$  \hfill (D.1)

where $Sd$ is the standard deviation and $N$ is the number of samples. In figures D1 and D2, we illustrate confidence intervals of 95% which correlate to a $z$-score of 1.96. Herein, we assume a Gaussian prior in our confidence interval calculations.
Figure B1. ASL alphabet difference confusion matrices.
Figure B2. ASL digits difference confusion matrices.
Table C1. ASL alphabet LeNet—varying parameter size comparison on Loihi.

| Model size | Parameters | Power (mW) | Latency (ms) | Energy (mJ) |
|------------|------------|------------|--------------|-------------|
| Small      | 22,870     | 35.64      | 12.11        | 0.432       |
| Medium     | 45,616     | 43.33      | 12.36        | 0.535       |
| Large      | 91,258     | 54.64      | 19.24        | 1.051       |

Figure D1. Confidence interval plot for different models on ASL alphabet dataset.

Figure D2. Confidence interval plot for different models on ASL digit dataset.

ORCID iDs

Mohammadreza Mohammadi https://orcid.org/0000-0002-1471-9871
Peyton Chandarana https://orcid.org/0000-0002-8945-8653
James Seekings https://orcid.org/0000-0002-4519-262X
Ramtin Zand https://orcid.org/0000-0002-1786-1152

References

[1] Cheok M J, Omar Z and Jaward M H 2019 Int. J. Mach. Learn. Cyber. 10131–53
[2] Tolentino L K S, Juan R O S, Juan A C, Thio-ac M A B, Pamahoy J R R, Forteza X J O and Garcia X J O 2019 Int. J. Mach. Learn. Comput. 9821–7
[3] Liao Y, Xiong P, Min W, Min W and Lu J 2019 IEEE Access 738044–54
[4] Wadhawan A and Kumar P 2021 Arch. Comput. Methods Eng. 28785–813
[5] Shukor A Z et al 2015 Proc. Comput. Sci. 7660–7
[6] Kumar P, Saini R, Behera S K, Dogra D P and Roy P P 2017 2017 15th IAPR Int. Conf. Machine Vision Applications (MVA) (Piscataway, NJ: IEEE) pp 157–60
