# Driving Scene Understanding using Spiking Neural Networks

by

Ramashish Gaurav

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2021

## Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Statement of Contributions

This thesis is based on the following two articles which I authored / co-authored:

1. Driving Scene Understanding: How much temporal context and spatial resolution is necessary? ***Ramashish Gaurav***, *Bryan Tripp*, and *Apurva Narayan*. Proceedings of the Canadian Conference on Artificial Intelligence - 2021 [32]. https://doi.org/10.21428/594757db.efb4fcff

2. Spiking Approximations of the MaxPooling Operation in Deep SNNs. ***Ramashish Gaurav***, *Bryan Tripp*, and *Apurva Narayan*. Under review at the Thirty-Sixth AAAI Conference on Artificial Intelligence (2022).

I contributed to the theory underlying my thesis and writing of the abovementioned papers with valuable inputs from my supervisors: Prof. Apurva Narayan and Prof. Bryan Tripp. I was solely responsible for the implementation and experimentation.

## Abstract

One of the applications of AI lies in developing intelligent systems for safe on-road driving, other than building and perfecting self-driving vehicles, and many others. Driving Scene Understanding (DSU) is one such area where AI algorithms can be used to infer the current actions of driver, pedestrians, nearby vehicles, etc. to improve the on-road decision making capability of the in-vehicle driver. Another related front of technological advancement in transportation is the production and development of electric vehicles. A future with battery electric vehicle and safe driving necessitates the creation of AI algorithms which not only assist in increasing the on-road safety but are also *energy efficient*.

This thesis is an attempt towards developing such an energy efficient AI model for DSU using Spiking Neural Networks (SNNs). Low power neuromorphic hardware (e.g. Intel's Loihi) can be leveraged for the deployment of such SNNs which offer low inference latency and energy efficiency. Out of a number of ways to build SNNs, an established method is to first train an Artificial Neural Network (ANN) with traditional neurons (e.g. ReLU) and then replace those neurons with spiking neurons (e.g. Integrate & Fire neurons) along with some other network modifications. Therefore, Chapter 4 first presents a 3D-CNN based ANN model, and identifies the appropriate spatial resolution and temporal depth of the incoming video frames for DSU. Through extensive experiments, it was found that MaxPooling performs better than AveragePooling in such a 3D-CNNs based model; however there exists no method to convert a network with MaxPooling layers into an SNN which can be entirely deployed on a specialized neuromorphic hardware.

Chapter 6 presents two novel approaches to implement MaxPooling on a neuromorphic hardware; thus facilitating the conversion of networks with MaxPooling layers to fully neuromorphic-hardware compatible SNNs. These approaches have been tested with 2D-CNNs based SNNs for image recognition, and can be extended to the 3D-CNNs based SNNs as well; thus, theoretically realizing an energy efficient SNN for DSU.

**Acknowledgements**

My interests in Computational Neuroscience begun since mid 2018, and I eventually found this Master's position at the University of Waterloo (UW) in January 2020. I express my heartful thanks to Prof. Apurva Narayan and Prof. Bryan Tripp for granting me this incredible opportunity which has set out a stepping stone towards my research career in Neuromorphic Computing domain. They have taught me the practice of pushing the limits to perfection while addressing research problems. Along with their valuable supervision, I am grateful for their constant encouragement and guidance throughout my program.

I am indebted to my parents for their unwavering support throughout my life so far. Their teachings and moral support have no equivalence. Among the well-wishers and kind people in my life, I would like to pay special thanks to Yash Mehrotra and family for their generosity and altruism. Thank-you all for being there in the crucial times of my life.

I am thankful to my MASc. committee members: Prof. Britt Anderson and Prof. John S. Zelek for sparing their valuable time to review this thesis and for their inputs. I also express my deep sense of gratitude to my friends for their love and support in numerous ways. Thank-you Mohammed Adnan, Anna Murynka, Davejot Sandhu, and Nolan Dey for their fruitful discussions around various research topics and always finding out time to accommodate my cravings for pizza, dinners, and movies. Thank-you Parag Yadav, Abha Jadon, Dolly Bajaj, Lakshit Battala, and Kritika Sharma who have always showered me with their love and support. Thank you Xueyang Yao, Narsimha Chilkuri, Parsa Torabian, Rajan Iyengar, Salman Khan, Sahand Shaghaghi, Peter Duggins, Kinjal Patel, and Natarajan Vaidyanathan for the fun times we had. The prolific discussions about the intricacies of Nengo ecosystem with the Nengo community members, especially with Eric Hunsberger, Xuan Choo, and Kinjal Patel call for a huge appreciation, thank-you so much.

With my Master's journey at the UW coming to an end, I have learned many valuable life lessons and research skills. I am thankful to all of the abovementioned people (and more) who contributed to keeping my life normal in these troubled times of the pandemic. Lastly, thank you mother nature for keeping me in good health and this opportunity.

## Dedication

*To the universe, where I am just a cog in the wheel.*

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

As per WHO [83], the number of deaths due to traffic accidents has risen steadily from 1.15 million in year 2000 to 1.35 million in year 2016; more people die due to road traffic injuries than from HIV/AIDS, tuberculosis or diarrhoeal diseases. Another WHO review [81] grimly reports that between 20 and 50 million people sustain non-fatal injuries in road accidents, in addition to the death of approximately 1.3 million people each year. This urgently calls for the development of AI systems (preferably in-vehicle deployable) which can mitigate road traffic accidents and increase on-road safety. Driving Scene Understanding (DSU) is one such avenue where AI algorithms can be used to infer the on-road driving scenarios in real-time, e.g. identification of taking turns, lane change, stop, etc.; thereby helping the in-vehicle driver (and/or nearby vehicles) to take quick and informed driving decisions.

However, the usage of current generation power-hungry AI algorithms for DSU is not the optimal way for the future ahead of us. Training the computationally expensive large AI models have a large carbon footprint [35]. Some studies report a release of up to or even more than 78,000 pounds of emissions [3] or even 626,000 pounds of carbon dioxide emissions [106, 70]. Accordingly, these AI models consume significant energy during inference mode as well. With the projected rise in the sale of electric vehicles (30% electric vehicles of all passenger cars by the year 2032) [93] , there arises a need to develop energy efficient AI models to prolong the battery life. Overall, the need of AI algorithms to increase the on-road driving safety in combination with their requirement to be energy efficient for their optimal deployment in electric vehicles, constitutes the motivation behind this thesis.

## 1.1 What are Spiking Neural Networks?

Our brain consumes 20W of power; and as mentioned above, present-day AI systems which try to imitate our brain end up consuming millions of Joules of energy while being trained and deployed, despite efforts being made to lower down the energy consumption (Fig. 5 in [114]). The computational units of the brain, i.e. our neurons encode and process information in terms of action potentials, informally called *spikes*. The highly optimized anatomy of the brain and the sparse spiking enables it to process information by consuming just 20W of power or less, whereas, a cortical simulation on IBM Blue Gene/P supercomputer consumes 2.9MW of power [37]. Spiking neurons mathematically model the physiological behavior of biological neurons, and the network built out of them closely imitates the information processing in the human brain. Such a network of spiking neurons is called Spiking Neural Networks (SNN) - the next ($3^{rd}$) generation of neural networks; consequently, they offer a promise of low power AI. A comparison between the energy consumption of SNNs and Deep Neural Networks (DNNs) is provided in the section 2.3.1. A short one minute video tour encapsulating the above paragraphs is recommended here (video submitted by me to the UW GRADflix 2021 competition).

Following are the five ways to build SNNs for deep learning (paraphrased from [87]).

1. Binarization of ANNs - traditional ANNs are trained with binary activation functions, information processing is still synchronous [44, 52, 92]
2. Conversion from ANNs - traditional ANNs are first trained with back-propagation, followed by replacing the analog/rate neurons with spiking neurons [24, 40, 98, 94]
3. Constrained training of ANNs - traditional ANNs are trained together with the constraints accounting for the properties of spiking neurons [27, 45, 46]
4. Supervised training of SNNs - directly training SNNs with spikes using variations of back-propagation e.g. using surrogate gradient descent techniques [14, 96, 72, 59, 79]
5. Unsupervised training of SNNs - directly training SNNs using STDP [10, 102] to effect a more biologically realistic training [51, 109]

In this work, I leverage the "Conversion from ANNs" (a.k.a. ANN-to-SNN conversion) method to build SNNs for DSU. For the same, a 3D-CNNs based ANN is first trained (for DSU task) with ReLU neurons using the conventional error back-propagation algorithm and then converted to an SNN using the NengoDL library [91].

## 1.2 Thesis Contributions

Contributions of this work can be summarized in the following points:

1. Presentation of a 3D-CNNs based network for DSU with a new state-of-the-art results on Honda Research Institute Driving Dataset (HDD). This contribution shows the advantage of 3D-CNNs over the prevalent Conv+LSTM based architectures for DSU.

2. Identification of appropriate spatial resolution and temporal depth of the input video frames for DSU with respect to a variety of driving situations. This analysis for DSU is first of its kind, providing insights into the spatial resolution and temporal depth parameters with which one can begin the experiments.

3. Introduction of a new accuracy metric **ASiST@x** to evaluate the performance of a model for continuous activity/scene recognition (applicable to DSU as well). Current metrics e.g. Average Precision do not convey any information about when a particular driving scene was recognized after its true transition; **ASiST@x** fills this gap.

4. Presentation of two novel methods to implement spiking-MaxPooling on a neuromorphic hardware (here Intel's Loihi chip). Currently, none of the MaxPooling methods for SNNs are deployable on a neuromorphic hardware; these two being the first.

5. Demonstration of the deployment of an SNN (for image recognition) with MaxPooling layers on a neuromorphic hardware (here Loihi boards). No SNN with MaxPooling layers have been deployed on a neuromorphic hardware so far; this being the first.

6. Presentation of a 3D-CNNs based SNN for DSU. All the five contributions above together culminate into designing of an entirely spiking 3D-CNNs based SNN for DSU, which is theoretically entirely deployable on Loihi boards but couldn't be evaluated (reasons described in Chapter 7).

## 1.3 Thesis Organization

Chapter 2 next presents the necessary background to review this thesis. Chapter 3 presents a prologue to the Chapter 4 which outlines the 3D-CNNs based ANN for DSU and other details. Chapter 5 then presents a prologue to the Chapter 6 which discusses about two novel methods for spiking-MaxPooling. Chapter 7 then combines the learnings from Chapter 4 and 6 to present a 3D-CNNs based SNN with spiking-MaxPooling layers for DSU. Chapter 8 then finally concludes this thesis with the direction for future work.

# Chapter 2

# Background Review

This chapter presents the basic details to review this thesis. In accordance with the ANN-to-SNN conversion method for building SNNs, I will start with the specifics for building a 3D-CNN based ANN, followed by the details of its conversion to an SNN; herein I will briefly introduce Nengo. I will then talk about the Neuromorphic Hardware and the Nengo ecosystem.

## 2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired from the biological neural networks in the brain. They constitute of weighted interconnected layers of neurons through which non-linear functions get computed. They can be considered as universal function approximators [84, 5, 42], where the input is fed-forward through the layers until the final output layer. Various types of layers constituting a typical ANN are described below.

### 2.1.1 Convolutional Layer

Convolutional layer is composed of locally connected weights (also known as kernels or filters) which are convolved over the patches of input data (or the previous layer's activation values). These kernels operate on locally present features, and convolution of such kernels over the entirety of input/intermediate activations maps attempts to determine the presence of those features at different locations. The weights of the kernels are learned as

part of the error back-propagation algorithm. During convolutions, the kernel weights are kept fixed and shared across the different locations in the input/intermediate-activations.

### 2D-Convolutional Layer

2D-Convolutional layer is designed for working with 2D spatial inputs e.g. images. The kernels represent features in two spatial dimensions, namely height/rows and width/columns. Consider a 2D input (image or activation map from the previous layer) $I$ of shape $(R, C, N)$ where $R, C$, and $N$ denote the number of rows, columns, and channels in the input $I$; $r \in [0, R), c \in [0, C)$, and $n \in [0, N)$. Let $K$ be a kernel bank of shape $(K_R, K_C, K_N, K_K)$ where $K_R, K_C, K_N$, and $K_K$ denote the number of rows, columns, channels, and number of kernels respectively in the kernel bank; $k_r \in [0, K_R), k_c \in [0, K_C), k_n \in [0, K_N)$, and $k_k \in [0, K_K)$. Let $bias_{k_k} \in \mathbb{R}$ denote the bias value associated with the kernel $k_k$. Note that $K_N = N$. Following Eq. 2.1 describes the 2D convolution operation:

$$O(r, c, k_k) = \sum_{k_r=0}^{K_R-1} \sum_{k_c=0}^{K_C-1} \sum_{k_n=0}^{K_N-1} ( I(r + k_r, c + k_c, k_n) \times K(K_R - k_r - 1, K_C - k_c - 1, k_n, k_k)) + bias_{k_k} \tag{2.1}$$

where $O(r, c, k_k)$ is the output after convolution operation. Note that the kernel $K$ is flipped while convolving, however, this is not required during implementation because flipping has no effect on the learning process in the network.

### 3D-Convolutional Layer

3D-Convolutional layer is designed for working with 3D spatial and temporal inputs; two dimensions denote the spatial aspects and the remaining one dimension denote the temporal aspects. However, sometimes 3D-Convolution is also used for 3D all-spatial inputs, e.g. medical images, fMRI scans, etc. Consider a 3D input (spatiotemporal video or intermediate activation maps) $I$ of shape $(T, R, C, N)$, i.e. an extra temporal dimension $T$ prepended to $(R, C, N)$ where $T$ denotes the number of temporal frames, $t \in [0, T)$, rest of the definitions same as in the previous section. Similarly, let $K$ be a kernel bank of shape $(K_T, K_R, K_C, K_N, K_K)$ where $K_T$ denotes the number of temporal frames considered in the kernel $K$, $k_t \in [0, K_T)$, rest of the definitions same as in the previous section.

Let $bias_{k_k} \in \mathbb{R}$ denote the bias value associated with the kernel $k_k$. Here too $K_N = N$. Following Eq. 2.2 describes the 3D convolution operation:

$$\begin{aligned} O(t, r, c, k_k) = \sum_{k_t=0}^{K_T-1} \sum_{k_r=0}^{K_R-1} \sum_{k_c=0}^{K_C-1} \sum_{k_n=0}^{K_N-1} ( \\ I(t + k_t, r + k_r, c + k_c, k_n) \times \\ K(K_T - k_t - 1, K_R - k_r - 1, K_C - k_c - 1, k_n, k_k)) + bias_{k_k} \end{aligned} \tag{2.2}$$

where $O(t, r, c, k_k)$ is the output after convolution operation.

### 2.1.2 Nonlinear Layer

This layer introduces nonlinearity in the ANNs, absence of which would result in ANNs being a cascading sequence of linear transformations only. Such a layer generally follows a weighted layer e.g. convolutional layer, and are correspondent to the layer of neurons. Most commonly used nonlinearity is the `ReLU(.)` activation function. Mathematically:

$$\texttt{ReLU}(x) = max(x, 0) \qquad \text{where } x \in \mathbb{R} \tag{2.3}$$

### 2.1.3 Pooling Layer

Pooling operation is typically used to downsample the intermediate feature maps. In addition, it also introduces translational invariance into the feature maps. Translational invariance means that the pooled output mostly remains unchanged upon small spatial translations of the input (or features in the input). Two pooling operations widely used are: AveragePooling and MaxPooling, generally applied next to the nonlinearity layer.

#### AveragePooling

AveragePooling takes an average of the inputs in a pooling window of a given dimension, e.g. 2D. It has the effect of smoothing out the intermediate feature maps. Mathematically, it can be written as follows ($P_R, P_C$ - are the 2D pooling window's rows and columns dimensions, $p_r \in [0, P_R), p_c \in [0, P_C)$, refer section 2.1.1 for rest of the symbols):

$$O(r, c, k_k) = \underset{p_r, p_c=0}{\overset{P_R, P_C}{\text{mean}}} I(r + p_r, c + p_c, k_k) \tag{2.4}$$

Note that in AveragePooling, all the values in a pooling window have a contribution.

#### MaxPooling

MaxPooling takes a max of the inputs in a pooling window of a given dimension, e.g. 2D. It is useful in detecting sharp features (e.g. edges), where AveragePooling might falter. Mathematically, it can be written as follows ($P_R, P_C$ - are the 2D pooling window's rows and columns dimensions, $p_r \in [0, P_R), p_c \in [0, P_C)$, refer section 2.1.1 for rest of the symbols):

$$O(r, c, k_k) = \underset{p_r, p_c=0}{\overset{P_R, P_C}{\max}} I(r + p_r, c + p_c, k_k) \tag{2.5}$$

Note that in MaxPooling, only the maximum value in a pooling window has a contribution.

### 2.1.4 Dropout Layer

Dropout layer is added as a simple and effective regularization technique to prevent overfitting of the ANNs. Based on a set probability $p_d$, it randomly disables those many neurons in proportion and scales the output of the rest by $\frac{1}{1-p_d}$. This helps keep the expected sum of the weights same regardless of dropout and enables the usage of the same network for training and test. Note that during test/inference, dropout is not used. Randomly disabling the neurons helps in the prevention of learning correlated features, and emphasizes on learning of discriminatory features which independently contribute to the task's objective.

### 2.1.5 Dense Layer

Dense layer is a fully connected layer of neurons and is commonly used in ANNs. Each neuron in a dense layer is connected to every neuron in the previous layer through weighted connections. The weighted sum with bias added is fed to the neuron's nonlinear activation function (section 2.1.2) and the output is forwarded next. Consider $W \in \mathbb{R}^{m \times n}$ as the matrix of connection weights from a previous layer with $m$ neurons to the current layer with $n$ neurons, $b \in \mathbb{R}^n$ as the vector of neurons' biases in the current layer, and $I \in \mathbb{R}^m$

as the vector of inputs (or activations) from the previous layer; then mathematically, the operation executed by the Dense layer can be written as follows (Eq. 2.6):

$$O = \sigma(I \times W + b) \tag{2.6}$$

where $\sigma$ is the nonlinear activation function e.g. `ReLU` and $O \in \mathbb{R}^n$ is the vector of activation output from the current layer's neurons.

### 2.1.6 Learning Network Parameters

Upon creation of an ANN model, its parameters i.e. weights ($W$) and biases ($b$) are randomly initialized as per a chosen distribution, e.g. $W \sim \mathcal{N}(\mu = 0, \sigma = 0.05)$, $b = \vec{0}$. While training the model, they are learned via an iterative gradient descent algorithm - an optimization algorithm to decrease the error between the true values (obtained beforehand) and the estimated values (obtained from the yet unoptimized model); the error is modeled as a differentiable loss function $\sim \mathcal{L}(Y, \hat{Y})$. $Y$ is the true/target output and $\hat{Y}$ is the estimated output, $\mathcal{L}(.)$ is the loss function e.g. Categorical Cross Entropy (Eq. 2.7):

$$\mathcal{L}(.) = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{C} y_s(i) \times log(\hat{y}_s(i)) \tag{2.7}$$

for $C$ number of classes, $S$ number of samples, $y_s \in Y$ and $\hat{y}_s \in \hat{Y}$ are the true/target and predicted vector outputs for the $s^{th}$ sample, $y_s(i)$ and $\hat{y}_s(i)$ are the $i^{th}$ element of the vectors $y_s$ and $\hat{y}_s$ respectively. The aim is to bring the values of vector $\hat{y}_s$ as close as possible to the values of vector $y_s$, for which one needs to minimize the loss function $\mathcal{L}(.)$. Loss function minimization is done by iteratively calculating the gradient of $\mathcal{L}(.)$ with respect to $W$ and $b$ and updating $W$ and $b$ as per the Eq. 2.8 and Eq. 2.9 respectively; note that $\hat{Y} = f(W, b, I)$ where $I$ is the input to the model.

$$W \leftarrow W - \eta\frac{\partial \mathcal{L}}{\partial W} \tag{2.8}$$

$$b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b} \tag{2.9}$$

In the above equations 2.8 and 2.9, $\eta$ is the learning rate - a tunable hyper-parameter.

## 2.2 Spiking Neural Networks (SNNs)

As described in section 2.2, SNNs are biologically inspired neural networks composed of spiking neurons which generate discrete spikes at certain time-steps based on their voltage dynamics; thus, the SNNs are inherently dynamical in nature. For each new static input, they are executed for a certain number of time-steps to let the neuron dynamics evolve (by keeping the static input fixed for those many number of time-steps). For a dynamic input, the SNNs dynamics evolve as the discretized time progresses, which makes it very suitable for addressing online tasks. The information processing in SNNs is sparse, asynchronous, and parallel.

There are a number of spiking neuron models with varying dynamics' complexity based on their biological realism. Hodgkin-Huxley neuron model [38] is a conductance-based spatial neuron model which governs the propagation of action potentials (i.e. spikes) along the axon through mathematical equations of voltage-gated ionic channels; thus, it has a high degree of complexity in its voltage dynamics. A few other neuron models similar to Hodgkin-Huxley neuron model but with comparatively simpler dynamics can be found in [113, 4]. Other examples of point neuron models (i.e. no spatial component) are Integrate & Fire (IF) neuron model [58] and Leaky Integrate & Fire (LIF) neuron model [58, 112] described below.

### 2.2.1 Integrate & Fire (IF) Neuron Model

IF neuron model is a spiking neuron model whose membrane potential (or the voltage $V(t)$) evolves by the following equation:

$$C_m \frac{dV(t)}{dt} = I(t) \quad \text{when} \quad V(t) < V_{th} \tag{2.10}$$

$$V(t) \leftarrow 0 \quad \text{when} \quad V(t) \geq V_{th} \quad \text{for } t_{th} \leq t \leq t_{th} + \tau_{ref} \tag{2.11}$$

where $C_m$ is the membrane capacitance, $I(t)$ is the external input current, $V_{th}$ is the threshold potential, $\tau_{ref}$ is the refractory period, and $t_{th}$ is the time-step when $V(t)$ reaches or crosses the $V_{th}$. For time-steps when $V(t) < V_{th}$, Eq. 2.10 governs the voltage dynamics, and as soon as $V(t)$ reaches or crosses the $V_{th}$, the IF model emits a spike and $V(t)$ is reset to 0 - and stays at 0 i.e. the resting potential (Eq. 2.11) for $\tau_{ref}$ period of time-steps.

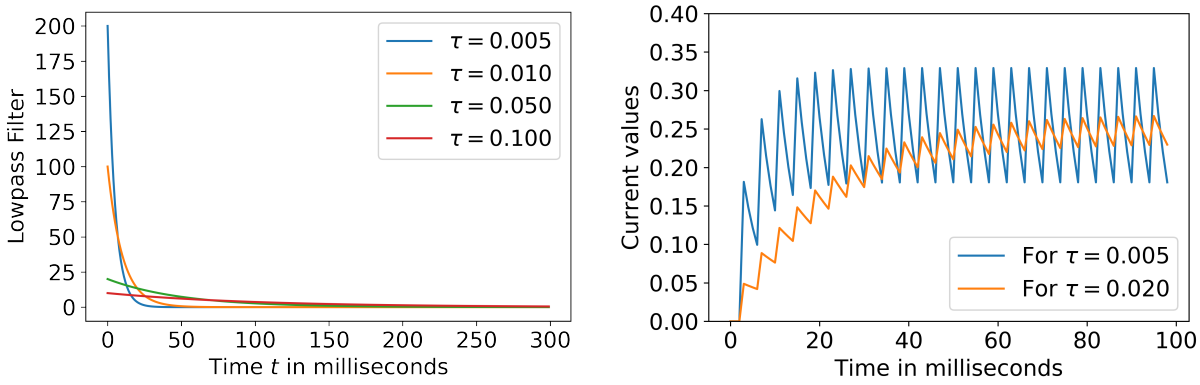
### 2.2.2 Leaky Integrate & Fire (LIF) Neuron Model

LIF neuron model is another spiking neuron model whose membrane potential (or the voltage $V(t)$) evolves by the following equations:

$$C_m \frac{dV(t)}{dt} = \frac{-1}{R_m}(V(t) - V_{rest}) + I(t) \quad \text{when} \quad V(t) < V_{th} \tag{2.12}$$

$$V(t) \leftarrow 0 \quad \text{when} \quad V(t) \geq V_{th} \quad \text{for } t_{th} \leq t \leq t_{th} + \tau_{ref} \tag{2.13}$$

where $R_m$ and $C_m$ are the membrane resistance and capacitance, $V_{rest}$ is the resting potential (generally assumed to be 0), $V_{th}$ is the threshold potential, $I(t)$ is external input current (due to the post-synaptic current and injected current - if any), $\tau_{ref}$ is refractory period, and $t_{th}$ is the time-step when $V(t)$ reaches or crosses the $V_{th}$. Comparing Eq. 2.10 and Eq. 2.12, an extra term $\frac{-1}{R_m}(V(t) - V_{rest})$ in Eq. 2.12 can be easily noted; this extra term is the leak current i.e. loss/diffusion of ions through the membrane, hence the name "Leaky" Integrate and Fire. For time-steps when $V(t) < V_{th}$, Eq. 2.12 governs the voltage dynamics, and as soon as $V(t)$ reaches or crosses the $V_{th}$, the LIF model emits a spike and $V(t)$ is reset to 0 - and stays at 0 i.e. the resting potential (Eq. 2.13) for $\tau_{ref}$ period of time-steps.




(a) Low-pass filter plots

(b) Representing activation as current

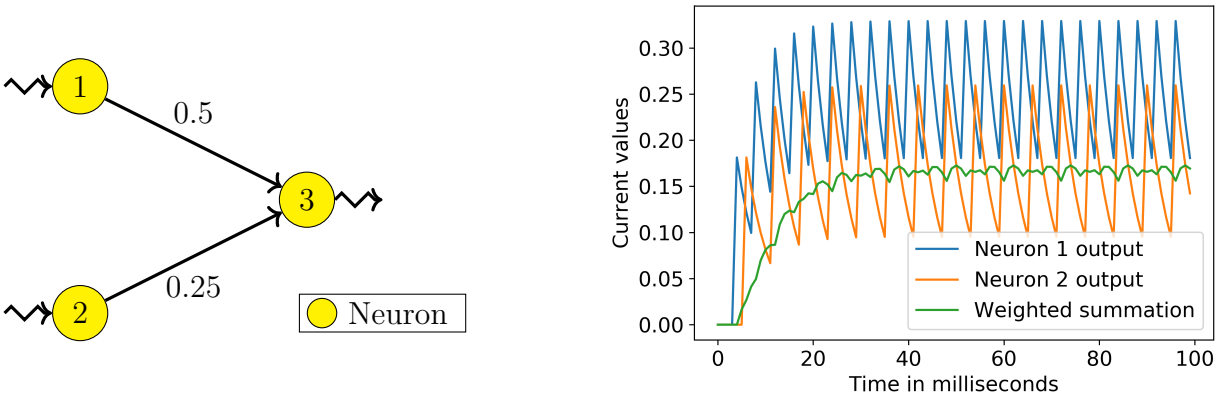
Figure 2.1: (a) shows the low-pass filter plots of Eq. 2.14 for different values of $\tau$, (b) shows the noisy current-based representation of the activation value 0.25 for different values of the synaptic time constant $\tau \in \{0.005, 0.020\}$.

### 2.2.3 Representing Activation Values as Currents in SNNs

As described in above sections, spiking neuron models emit discrete spikes, effectively a spike train - a vector of 1s and 0s when simulated over certain number of time-steps. $1 \implies$ spike emitted in a particular time-step, $0 \implies$ no spike emitted i.e. the neuron's $V(t)$ is either building up or it is in refractory period during those time-steps. When leveraging the ANN-to-SNN conversion method for building SNNs, we need to represent the continuous activation values (in ANNs) in SNNs as well. We can do that by synapsing/filtering the spike train with a low pass filter - the resulting signal (a.k.a. post-synaptic current) is noisy, but approximately represents the continuous activation values. Following Eq. 2.14 describes impulse response function of a standard low pass filter:

$$f(t) = \frac{1}{\tau}\exp\left(\frac{-t}{\tau}\right) \tag{2.14}$$

where $\tau$ is the time constant of the filter. Fig. 2.1a shows the synaptic filter's plot (Eq. 2.14) with respect to time and varying time-constants $\tau$. Fig. 2.1b shows the representation of continuous-valued activations through a (noisy) current obtained by synapsing/filtering a spike train through a low-pass filter with $\tau = 0.005$. Note that as the filter's time-constant $\tau$ is increased, the post-synaptic current gets smoother, but takes longer to settle.




(a) Example spiking network

(b) Weighted summation result

Figure 2.2: (a) shows a spiking network of three neurons. Neuron 1 and 2 output spikes which are filtered and subsequently weighted with the connection weights 0.5 and 0.25 respectively. The weighted signals get summed up while being input to Neuron 3. (b) shows the weighted summation of two input signals to Neuron 3.

### 2.2.4 Weighted Transformations in SNNs

Other than representing activations through spiking activity, another integral aspect of ANNs that needs to be accounted in SNNs is the weighted summation of output activations. This can be done by setting the weights on the connections between spiking neurons, as shown in Fig. 2.2a. The resulting output signal (i.e. the weighted sum of input activations) can be fed forward to the next connected spiking neuron. By means of such weighted linear-transformations and non-linear spiking activity of the neurons, non-linear functions get computed through the cascading network of spiking neurons. Fig. 2.2b shows the output signal obtained from the example network shown in Fig. 2.2a.

## 2.3 Neuromorphic Hardware

To reap the full potential of SNNs, we need Neuromorphic Hardware as a substrate for their deployment. Traditional von Neumann architecture (Fig. 2.3) has a clear separation of the computing unit (ALU) from memory, which introduces latency in fetching stored values for computation. This further manifests into von Neumann bottleneck (a.k.a. memory bottleneck) where processing of the computing tasks is hindered due to the bandwidth limitation of memory access. The CPU often does the computation on data faster and then has to wait for the next batch of incoming data. Thus, much energy is spent on moving the data across the processors and memory through buses, than actually spent on doing the computations with it. Despite improvements made by introducing cache, non-uniform memory access (NUMA) architecture, parallel processing in GPUs, etc., execution of AI algorithms on von Neumann based architectures is inefficient due to high energy consumption and latency [49, 114], irrespective of being in training or inference mode.

Neuromorphic Hardware is based on the brain-inspired architecture where synapse (or memory) and compute are local to each other [68, 78]; this enables such a non-von Neumann hardware to avoid memory bottleneck and carry out massively parallel computations in an asynchronous event-based manner. Such local arrangement also dramatically reduces the need to move data around (note: decreased latency); thus, Neuromorphic Hardware is highly energy efficient [108, 78]. Fig. 2.4 shows an illustration of a basic concept of a neuromorphic architecture (adapted from [76]). Neuromorphic Hardware owes its energy efficiency to the non-volatile memory devices e.g. resistive RAM (RRAM) [33, 17], phase-change RAM (PCRAM), conductive-bridge RAM (CBRAM) [55], OxRAM [13], etc.

A number of neuromorphic hardware currently existing are: IBM TrueNorth chip [73], Stanford NeuroGrid [7], Braindrop chip [77], Manchester SpiNNaker [30], Intel Loihi chip

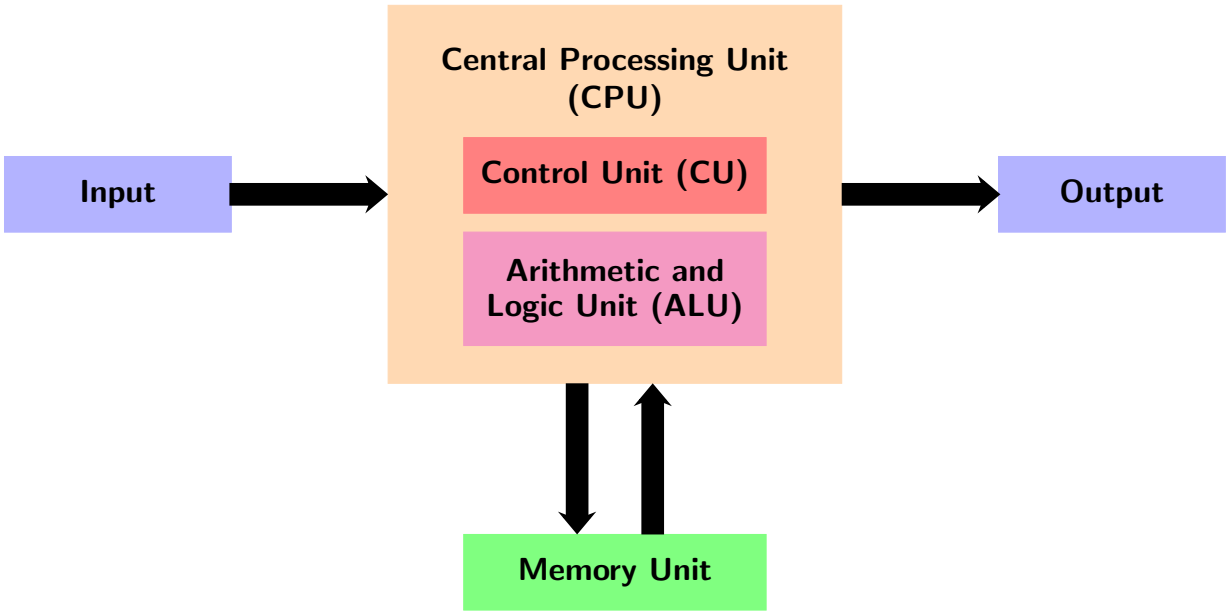



Figure 2.3: von Neumann Architecture

[21], Darwin Neural Processing Unit [101], ROLLS chip [88], DYNAPs [74], and Heidelberg BrainScaleS system which uses wafer-scale above-threshold analogue neural circuits [95].

### 2.3.1 Energy Efficiency

On one hand where the cortical simulation of 1.617 billion neurons and 8.87 trillion synapses on IBM Blue Gene/P supercomputer [2] with 147,456 CPUs and 144TB of main memory consumes 2.9MW of power [37], our brain with 86 billion neurons and 150 trillion synapses consumes just 20W of power [50]. Neuromorphic hardware consume energy in the orders of pico-joules (pJ) per spike [67, 90, 19, 71]. When put to real-world tasks, it was shown that Loihi (a neuromorphic chip) can classify odor samples within 3 ms of time with a consumption of less than 1 mJ of energy [47]. In a keyword spotting task, [12] reported 0.27 mJ of energy consumed per inference on Loihi compared to 29.8 mJ on GPU. Similarly, for image retrieval task, [66] reported a high energy consumption of 52.399 mJ (and 37.399 mJ) per example on NVIDIA V100 (and NVIDIA T4) GPU compared to 2.996 mJ (and 12.17 mJ) with SNNs on Loihi executed for 16 (and 128) time-steps. For image segmentation, [85] reported an energy consumption of 30 mJ on GPU and 10 mJ on Loihi per inference.

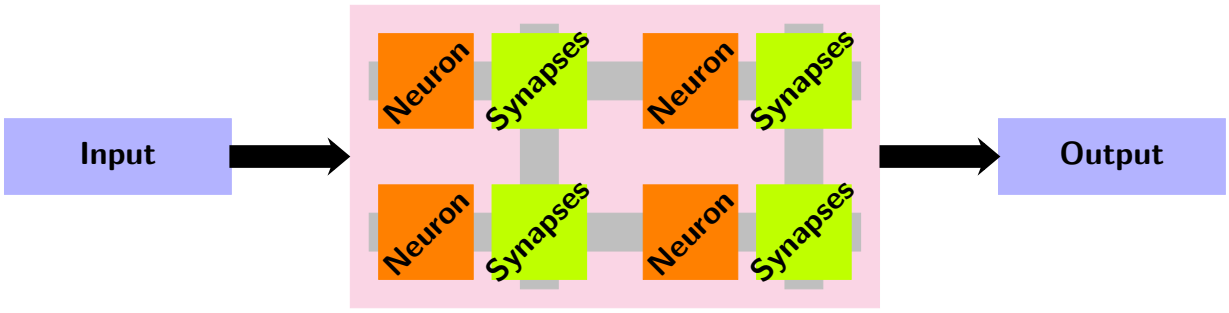



Figure 2.4: Basic abstraction of a non-von Neumann Neuromorphic Architecture

## 2.4 Nengo Ecosystem

Nengo ecosystem is a collection of python libraries (primarily based on Neural Engineering Framework [26, 103]) for building and simulating spiking networks - developed at Centre for Theoretical Neuroscience, University of Waterloo; and actively maintained by Applied Brain Research. Among the constituent libraries, following are the ones used for this work.

### 2.4.1 Nengo Core

Nengo Core [104, 6] is a foundational library in Nengo ecosystem for building and simulating large-scale spiking networks (or neural models) and underlies the other libraries. Networks built with Nengo Core can be deployed on CPUs/GPUs/Loihi/SpiNNaker/FPGA [9] depending on the choice of the simulator.

### 2.4.2 NengoDL

NengoDL is a simulator for executing deep learning based spiking networks (or SNNs) on CPUs/GPUs [91]. It uses TensorFlow in the backend to optimize the model parameters using standard deep learning algorithms and assists in ANN-to-SNN conversion (i.e. from TensorFlow non-spiking models to Nengo SNNs).

### 2.4.3 NengoLoihi

NengoLoihi is a backend simulator for executing spiking networks on Intel's Loihi boards. It uses the Intel's NxSDK API (proprietary to Intel) to communicate with the Loihi chips.

## 2.5 Driving Scene Understanding

Briefly introduced in Chapter 1, within the scope of this thesis, Driving Scene Understanding (DSU) can be defined as the *recognition of ongoing on-road driving scenarios* [89] as and when the input data arrives. In a broader scope, Driving Scene Understanding is different from the regular Scene Understanding; Driving Scene Understanding may include e.g. predicting the trajectory of on-road vehicles/pedestrians etc., whereas Scene Understanding encompasses the extraction and meaningful representation of the semantic information/relationship shared by the objects in a 3D dynamic scene [86], e.g. describing the activities happening in an elevator, generating meaningful text/audio to the visuals of an ongoing game, etc. Here, in the context of DSU, the on-road driving scenarios are e.g. the ego (short for *egocentric*) vehicle taking a left/right turn, ego vehicle changing lanes, ego vehicle coming to a stop due to the traffic lights, stop signs, or due to the on-road pedestrians, etc. (more scenarios in the Tables 4.2, 4.3, and 4.4). The task is to identify what's happening in the current scene of interest. The input data can be visual input (e.g. from mounted cameras), vehicle sensor readings (e.g. from speedometer, steering wheel angle, etc.), LiDAR data, etc. Due to the discrete nature of the computer programs, one needs to sample the input data at a certain rate e.g. videos sampled at 30FPS, sensor data sampled at 30Hz, and recognize an on-going driving scene right at the arrival of the input data (in the discretized time-step). To recognize scenes, AI models e.g. Conv+LSTM based architectures [89, 29, 110] are widely used and many others; this thesis focuses on the 3D-CNNs based models. More details on the DSU task can be found in section 4.1.1.

## 2.6 Conclusion

This chapter introduced the basics to understand the foundations of this thesis. With the goal of building a Spiking Neural Network for Driving Scene Understanding using the ANN-to-SNN conversion method, this chapter first presented the building blocks of ANNs (CNNs in this thesis' case) and training them in section 2.1, followed by the building blocks of SNNs in section 2.2. In accordance with the goal to also deploy the SNNs on neuromorphic hardware, section 2.3 briefly introduced and discussed its architecture. Section 2.4 then next introduced the Nengo ecosystem of different libraries to build, simulate, and deploy the SNNs on a neuromorphic hardware (here Intel's Loihi). Section 2.5 finally described the Driving Scene Understanding task in this thesis' scope.

# Chapter 3

# Foreword to Article 1 in Chapter 4

The goal of this thesis is to build a Spiking Neural Network for Driving Scene Understanding using the ANN-to-SNN conversion method. Therefore, to build a suitable ANN first, Article 1 in Chapter 4 presents the rationale behind the choice of the ANN, appropriate configuration (i.e. metadata) of the input data, and the evaluation metric. Learnings from the Chapter 4 will be useful later to build the desired SNN.

## Foreword

Driving Scene Understanding is a broad field which addresses the problem of recognizing a variety of on-road situations; namely driver behaviour/intention recognition, driver-action causal reasoning, pedestrians' and nearby vehicles' intention recognition, etc. Many existing works propose excellent AI based solutions to these interesting problems by leveraging visual data along with other modalities. However, very few researchers venture into determining the necessary metadata of the visual inputs to their models. Chapter 4 attempts to put forward some useful insights about the required spatial resolution and temporal context/depth of the visual data for Driving Scene Understanding (DSU). It also presents a 3D-CNNs based network for DSU with a new state-of-the-art results on the Honda Research Institute Driving Dataset (HDD) - on visual data alone. Further, this chapter introduces a new accuracy metric **ASiST@x** for measuring the network's performance on continuous scene/activity recognition.

# Chapter 4

# Driving Scene Understanding using 3D-CNNs based ANN

## 4.1 Introduction

With the ubiquitous use of AI in every walk of life, the transportation domain is not untouched by it; AI is increasingly finding its varied usage in autonomous and semi-autonomous vehicles. In a future of hybrid transportation where more and more of human-driven vehicles and self-driven vehicles would share the road, there arises a need to recognize a driving scene for better contextual communication among vehicles to make informed driving decisions, thereby, increasing the on-road safety. Therefore, researchers employ a variety of modalities e.g. visual data, Controller Area Network (CAN) bus data, LiDAR data, GPS data, etc. to understand a driving scene, with the visual data being the primary modality. This work has been published at the Canadian Conference on Artificial Intelligence - 2021 [32].

### 4.1.1 Driving Scene Understanding

In the context of Driving Scene Understanding, a number of works have been done; each of which addresses different subsets of problems in this broad domain. Authors in [8, 39, 25] leverage Hidden Markov Models for driver intention recognition. Recently, Casas et al. [16] put forward a fully convolutional neural network method for predicting the driving intent of other vehicles in the context of self-driving ones. For the task at hand, they

leveraged the 3D point clouds produced by the mounted LiDAR and dynamic maps of environment containing lanes, intersections, etc. Frossard et al. [29] proposed the usage of a Convolutional-Recurrent architecture for detecting the turn signals and flashers in video sequences. A few [97, 11] have also attempted to understand and predict the pedestrian intentions to improve the Driver Intention Recognition systems. Torstensson et al. [110] proposed a Convolutional and LSTM based network to predict the actions of the in-vehicle driver. In a work related to Driving Action Anticipation, Aliakbarian et al. [1] introduced a new dataset: VIENA$^2$ and proposed a multi-modal LSTM based network to forecast driver actions from visual and sensor data.

A recently published dataset by Ramanishka et al. [89]: **Honda Research Institute Driving Dataset** (**HDD**) which they benchmark for a variety of driving scenes, has gained traction of late for the task of Driving Scene Understanding due to its rich annotations and temporally aligned visual-sensor data. Xu et al. [116] used this dataset to evaluate their new recurrent architecture for a variety of online action-detection tasks (including driving scenes). The authors in both the papers [89, 116] used a Convolutional network coupled with a Recurrent architecture for recognizing the driver actions from visual as well as the CAN bus sensor data. The HDD dataset was further studied by Li et al. [61] to identify the causal reasons for the human drivers to stop on-road. This dataset was also employed for interaction modeling between ego-car and other on-road objects (e.g. pedestrians, lanes, traffic light) by using Graph Convolutional Networks [62]. Owing to the popularity and variety, we use the HDD dataset along with VIENA$^2$ for our experiments.

### 4.1.2 3D-CNNs

Visual data forms an important modality for Driving Scene Understanding and Convolutional networks are critical to learning spatial representations. While most of the works use pre-trained 2D-CNNs (primarily on ImageNet) to extract spatial features, followed by the usage of Recurrent networks for learning temporal dynamics; researchers have largely ignored 3D-CNNs based models for Driving Scene Understanding. 3D-CNNs based models can jointly learn the spatial and temporal representations in a video [111, 23, 22] and can also be used for human action recognition [48]. Another missing aspect of most of the works is the absence of insights in the necessary spatial and temporal resolution of visual inputs for Driving Scene Understanding. One can expect higher spatial resolution to be favourable performance-wise; however, it comes with an added cost of increased computational complexity, thus, high power requirements. Increased prior temporal context to understand an ongoing driving scene might introduce irrelevant past contextual details (e.g. lane changes are quicker than U-turns). In addition, the joint contribution of higher

spatial resolution and increased temporal context also poses hardware implications during training and deployment.

Therefore, we build a Convolutional 3D (C3D) [111] inspired 3D-CNN based network and investigate the degree of the required spatial resolution and temporal depth for Driving Scene Understanding. C3D is one of the best established architectures for a variety of video based tasks and our improved results with it shows its efficacy. Since Tran et al. [111] already did exhaustive hyper-parameter search while building their C3D architecture, we reuse their findings in building our model and focus on the less studied spatial and temporal resolution need instead. We will revisit with a short discussion on the C3D architecture's hyper-parameters in Chapter 8 (Conclusion and Future Work).

### 4.1.3 Contributions

This chapter's contribution is three fold, summarised below.

- We demonstrate superior results on the visual data alone with our C3D inspired 3D-CNNs based architecture
- Given a model, we attempt to identify the optimum spatial resolution and temporal context/depth of the input necessary for recognizing a variety of driving scenes (collectively) in general setting
- We introduce a new accuracy metric **ASiST@x** which jointly measures the accuracy of recognizing scenes within a certain time as well as the recognition continuity of ongoing scenes

We organize this chapter as follows. In Section 4.2 we define the specifics of our experiments, followed by the experimental details in Section 4.3. We then present and analyse our results in Section 4.4 followed by consolidating our findings in the conclusion Section 4.5.

## 4.2 Experiment Specifics

In this section we formally describe the elements of our designed experiments. We begin by defining the term **Temporal Context/Depth**, followed by defining the term **Spatial Resolution**, and end this section with a description of our 3D-CNNs based model.

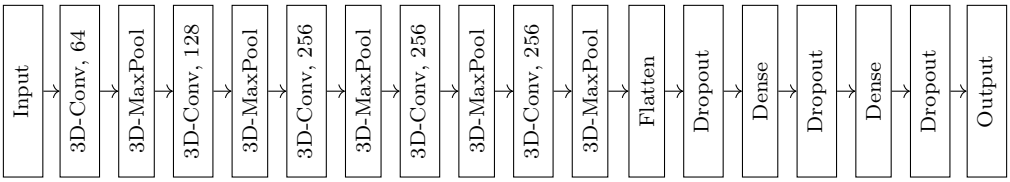



Figure 4.1: Our Model's Architecture; "3D-Conv, 64" implies 64 filters in the 3D-Convolutional layer

### 4.2.1 Temporal Context/Depth

The RGB video data is composed of a continuous sequence of frames $f_m$ where $m \in [1, \cdots, N]$; $N$ is the total number of frames in the video, usually at 30 FPS. We can label each of the frames $f_m$ sequentially to denote various temporally arranged ongoing driving scenes. Let $F_{i,j}$ be a contiguous sequence of consecutive frames $f_m$ where $m \in [i, \cdots, j]$. We therefore build a sliding window $F_{i,i+l-1}$ of $l$ frames where the task is to predict the label of the last frame $f_{i+l-1}$; the label denotes the ongoing driving scene in the current frame $f_{i+l-1}$ in the context of past $l-1$ frames. We slide the window one frame at a time. Contrary to [89, 116] where authors first construct a sequence of 90 frames and then predict the labels of each frame in one go, our formulation is more favourable and responsive to the time critical on-road situations, as it recognizes the ongoing driving scene immediately upon arrival of a new frame. We mention this parameter $l$ as the **Temporal Depth**. In our experiments, we consider three different values of $l \in \{16, 24, 32\}$; *what should be the optimum $l$?*

### 4.2.2 Spatial Resolution

Irrespective of the various Neural Network models to learn the spatial features, the question remains: *what should be the appropriate spatial resolution of the input frames?* Towards this cause, we resize the frames to varying spatial resolutions; low resolution: $36 \times 64$ pixels, medium resolution: $72 \times 128$ pixels, and high resolution: $108 \times 192$ pixels, and conduct extensive experiments. The first dimension corresponds to the number of rows and second dimension corresponds to the number of columns in the RGB frames.

### 4.2.3 3D-CNN based Model

To benchmark our findings we build our model (Figure 4.1) based on the C3D architecture proposed by Tran et al. [111]. Each of the Convolutional layers has a kernel size of $(3, 3, 3)$. The Max-Pooling size is set to $(2, 2, 2)$ except for the first pooling layer where it is set to $(1, 2, 2)$. The first dimension of the kernel size and pool size correspond to the temporal dimension, last two dimensions correspond to the spatial dimension. The strides for the Convolutional layers are set to $(1, 1, 1)$ and that for the Max-Pooling layers are set as the pool size. The number of neurons in each of the non-output Dense layers is set to 2048. All the neurons in our model are $ReLU$ neurons, except for the output layer which has softmax activation. To prevent overfitting, we $L2$ regularize the kernels and keep the dropout probability = 0.25. The learning rate is fixed at 0.0001 and we use the optimizer Adam [54] to train our network. In accordance with [89, 116] we also use Focal Loss [64] (with $\gamma = 2.0$) as the loss function to account for the class imbalance problem in the HDD dataset.

## 4.3 Experiments

In this section we describe the details of our conducted experiments. We begin by a short introduction of the VIENA$^2$ and the HDD dataset, followed by the experiment methodology. Note that we used only the visual data, and sampled the frames at 3 FPS for both datasets (authors in [89, 116, 62] sample the HDD dataset at 3 FPS).

### 4.3.1 VIENA$^2$ Dataset

The VIENA$^2$ dataset consists of multiple 5 seconds long labelled video clips (30 FPS, $1280 \times 1920$ pixels resolution, $\approx 8.5$ hours total) along with the aligned sensor data (speed and steering angle) collected from the GTA V video game for five different driving Scenarios; namely (1) Driver Maneuvers (DM $\approx 2h45m$), (2) Accidents (AC $\approx 1h25m$), (3) Traffic Rules (TR $\approx 1h30m$), (4) Pedestrian Intentions (PI $\approx 1h10m$), and (5) Front Car Intentions (FCI $\approx 1h45m$). For each of the 5 scenarios, there are 3 different splits: *Daytime* split, *Weather* split, and *Random* split; we use the *Random* split (70% training clips, 30% test clips). Note that due to only 5 seconds long clips, we could not experiment for $l = \{24, 32\}$.

| | HDD | HDD | VIENA$^2$ - *Random* split | | | | |
|---|---|---|---|---|---|---|---|
| | **Layer 0** | **Layer 1** | **Scenario 1** | **Scenario 2** | **Scenario 3** | **Scenario 4** | **Scenario 5** |
| **Training** | 113 | 105 | 3 | 2 | 2 | 2 | 2 |
| **Inference** | 79 | 82 | 2 | 1 | 1 | 1 | 1 |

Table 4.1: Average run-time (rounded) in minutes of $E_{108\times192,16}$ for the HDD and VIENA$^2$ dataset. Note that the training was done in parallel on 4 GPUs, but the inference was done on a single GPU.

### 4.3.2 Honda Research Institute Driving Dataset (HDD)

The HDD dataset [89] is a real world largest public dataset (till date [62]) containing 104 hours of egocentric driving data with per frame annotation. It has video data (30 FPS, $720 \times 1280$ pixels resolution) and CAN Bus sensor data for 137 driving sessions (avg. span 45 minutes); 100 sessions are used for training and rest for testing. This dataset has 4 "Layers" of annotation for Driving Scene Understanding; namely: Goal-Oriented Action, Stimulus-Driven Action, Cause, and Attention (authors in [89] use the term *"Layer"* to denote groups of semantically related driving scenes). We experiment on two annotation layers: **Layer 0 - Goal-Oriented Action** and **Layer 1 - Cause**. We do not use the CAN Bus sensor data.

### 4.3.3 Model Execution

We conduct a number of experiments with VIENA$^2$ (5 runs each) and both Layers of the HDD dataset (3 runs each), where each experiment corresponds to a combination of a spatial resolution and a temporal depth. Across all the experiments, the model hyper-parameters are kept constant for fair comparison; only the spatial resolution and the temporal depth is varied. For convenience, we mention each experiment as $E_{r\times c,l}$ where $r \times c \in \{36 \times 64, 72 \times 128, 108 \times 192\}$ pixels resolution, and $l \in \{16, 24, 32\}$ frames temporal depth. Thus, the shape of input to our model is $(batch\_size, l, r, c, 3)$ where 3 is the RGB channel dimension. The experiments are executed on nodes with 4 NVIDIA V100 32GB GPUs. For the HDD and VIENA$^2$ dataset, 1 epoch's training and inference run-time figures (averaged across runs) of the best performing experiment $E_{108\times192,16}$ are mentioned in Table 4.1. Since the authors [1, 89, 116, 62] (with whom we compare our results later) do not provide run-time estimates of their experiments, we are unable to present a comparison. Our code is publicly available [1].

[1] https://github.com/R-Gaurav/DSU-3D-CNNs

| | Scenario 1: DM | | | | | | Scenario 2: AC | | | | Scenario 3: TR | | | | | Scenario 4: PI | | | | Scenario 5: FCI | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **FF** | **SS** | **LL** | **RR** | **CL** | **CR** | **AC** | **AP** | **AA** | **NA** | **SR** | **PR** | **WD** | **CD** | **DO** | **CR** | **SS** | **AS** | **NP** | **FF** | **SS** | **LL** | **RR** | **CL** | **CR** |
| $E_{36\times64,16}$ | 72.9 | 95.4 | 88.1 | 88.2 | 59.7 | 76.3 | 85.9 | 68.7 | 41.3 | 96.2 | 98.7 | 46.4 | 58.0 | 71.6 | 45.1 | 53.3 | 39.6 | 62.1 | 60.0 | 80.1 | 84.4 | 64.7 | 66.4 | 31.0 | 1.8 |
| $E_{72\times128,16}$ | 83.2 | 95.7 | 84.7 | 83.9 | **68.9** | **82.9** | 81.0 | 55.4 | 50.8 | 95.9 | **100** | 50.4 | 57.7 | 69.0 | 36.9 | 52.1 | 36.8 | **75.1** | 67.0 | 80.4 | 88.6 | 70.8 | 66.4 | 36.8 | 0.0 |
| $E_{108\times192,16}$ | 79.4 | 95.2 | 76.5 | 72.8 | 65.1 | 76.6 | 73.6 | 70.3 | 45.1 | 92.6 | 99.0 | **54.6** | 49.3 | 68.6 | 41.5 | 68.3 | 26.8 | 71.2 | 58.5 | 71.3 | **92.8** | 69.5 | 69.2 | 47.7 | 1.8 |
| [1] | **88.0** | **97.2** | **95.8** | **90.4** | 64.9 | 65.4 | **86.1** | **80.5** | **80.2** | **100** | 95.1 | 40.0 | **75.0** | **85.7** | **48.6** | **78.2** | **76.6** | 63.6 | **74.1** | **91.2** | 83.5 | **84.6** | **81.4** | **59.4** | **66.8** |

Table 4.2: VIENA$^2$ accuracy scores for all 5 Scenarios - *Random* split; **DM** - Driver Maneuvers, **AC** - Accidents, **TR** - Traffic Rules, **PI** - Pedestrian Intentions, **FCI** - Front Car Intentions. For respective class acronyms (e.g. FF, SS, AP, NP, etc.) definition, refer Section 2.1 of [1].

## 4.4 Results & Analysis

Here, we present our results (averaged across runs) and analyse them to get insights in the optimal spatial and temporal resolution required for understanding a variety of driving scenes. We begin by proving the efficacy of 3D-CNNs over existing approaches, followed by analysing the per-frame **ASiST@x** plots of all $E_{r\times c,l}$ collectively on the HDD dataset.

### 4.4.1 VIENA$^2$ Dataset Results

We present our results for the VIENA$^2$ dataset in Table 4.2, where we compare our class-wise accuracy scores (obtained with $E_{r\times c,16}$) with that of Aliakbarian et al. [1] (obtained on the visual and sensor data, at the end of the $5^{th}$ second). In the experiments $E_{r\times c,16}$, we found that the class-wise accuracy scores for each Scenario plateaus after the $50^{th}$ epoch, with slight fluctuations later (total number of epochs run - 64). Since the motive of these experiments was to find perceptible differences in the performance of each spatial resolution, we chose not to report the highest class-wise accuracy scores obtained at different epochs; rather we report the results of the $64^{th}$ epoch for all of the $E_{r\times c,16}$ for a fair comparison. As can be seen in Table 4.2, scores across different $E_{r\times c,16}$ do not lead to a conclusive evidence about which spatial resolution is superior (we observed similar ambiguity with the highest class-wise accuracy scores too). We attribute this inconclusiveness to the small scale of the dataset; each Scenario is just 1 hour to 3 hours in total. However, it is observable that our 3 FPS visual only 3D-CNNs based model outperforms the 30 FPS multi-modal pre-trained CNN-LSTM based model for few classes, which shows its efficacy.

| | AP results: Layer 0 - Goal Oriented Action Layer | | | | | | | | | | | mAP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Methods** | **Right Turn** | **Itr. Pass.** | **Merge** | **Left Lane Change** | **Right Lane Branch** | **Right Lane Change** | **Left Turn** | **Crs. Pass.** | **Rail. Pass.** | **Left Lane Branch** | **U-Turn** | |
| [89] | 54.43 | 65.74 | 4.86 | 27.84 | 1.77 | 26.11 | 57.79 | **16.08** | 2.56 | 25.76 | 13.65 | 26.96 |
| [116] | 57.3 | 63.5 | 3.5 | 28.4 | **10.5** | 37.8 | 57.0 | 11.0 | 0.5 | 31.8 | **25.4** | 29.7 |
| [62] | **71.7** | 72.8 | 10.6 | 53.4 | 3.1 | 44.7 | 64.8 | 14.6 | **2.9** | **52.2** | 15.8 | 37.0 |
| **Ours** | 70.13 | **78.14** | **12.18** | **55.26** | 9.91 | **46.41** | **66.82** | 13.53 | 0.59 | 46.51 | 12.17 | **37.42** |

Table 4.3: Average Precision (AP) results for HDD Layer 0. **Itr. Pass.**: Intersection Passing; **Crs. Pass.**: Crosswalk Passing; **Rail. Pass.**: Railroad Passing; **mAP**: mean Average Precision

| | AP results: Layer 1 - Cause Layer | | | | | | mAP |
|---|---|---|---|---|---|---|---|
| **Methods** | **Congestion** | **Sign** | **Traffic Light** | **Crossing Vehicle** | **Parked Car** | **Pedestrian** | |
| [89] | 39.72 | 46.83 | 45.31 | NA | **7.24** | 2.15 | 28.25 |
| **Ours** | **76.84** | **47.19** | **67.70** | 17.42 | 2.29 | **4.54** | **39.71** |

Table 4.4: Average Precision (AP) results for HDD Layer 1. While calculating and comparing our **mAP**, AP of **Crossing Vehicle** is not accounted as it was not reported in [89]

### 4.4.2 HDD Dataset Results

Authors in [89, 116, 62] chose to report the Average Precision (AP) scores of each driving scene in a Layer; for comparison, we do the same. We executed the experiments $E_{r\times c,l}$ for both layers, Layer 0: Goal-Oriented Action and Layer 1: Cause, for varying number of epochs (10 to 16). Upon observing the inference mean Average Precision (mAP) scores at the end of each epoch we found that it plateaus (with minimal variations) after $6^{th}$ epoch in all the experiments. Therefore we present and analyse the AP, mAP, and ASiST@x scores obtained at the end of $7^{th}$ epoch (this also helps towards fair comparison of different $E_{r\times c,l}$).

#### AP and mAP Score Analysis

Tables 4.3 and 4.4 show the AP results of the experiment $E_{108\times 192,16}$ for Layer 0 and Layer 1 respectively. In Table 4.3, we see that our results vastly outperform the ones [89, 116]

obtained by the coupled Convolutional-Recurrent based models. Results of [62] are closer to ours because it corresponds to their *online* C3D framework. It is notable that they [62] obtained their results with input clips of 20 frames and a resolution of $224 \times 224$ pixels, whereas, our results with 16 frames clip and $\approx 60\%$ smaller resolution beats theirs in almost half of the driving scenes. In Table 4.4 we compare our visual only results with [89] obtained on the visual and CAN Bus sensor data (due to the absence of their results on visual data alone). Here also we note that our visual only model beats their multi-modal coupled Convolutional-Recurrent model; thus showing the efficacy of our 3D-CNNs based model to effectively capture the spatiotemporal features.

**Algorithm 1: ASiST@x** metric calculation

**Input :** $K$, Array $[l_i^t]$, and $[l_i^p]$
**Output:** $lmatched\_at\_x$, $fcount\_at\_x$

**Initialization:**
  $len\_l^t \leftarrow$ Length of $[l_i^t]$
  $x \leftarrow 0$ /* Relative index of next frame since a scene's transition */
  $lmatched\_at\_x \leftarrow [0, \cdots, 0]$ /* Array of zeros of length $K+1$ */
  $fcount\_at\_x \leftarrow [0, \cdots, 0]$ /* Array of zeros of length $K+1$ */
**for** $i \leftarrow 1$ **to** $len\_l^t$ **do**
  **if** $l_i^t \neq l_{i-1}^t$ **then**
    /* Scene transition detected */
    $x \leftarrow 0$ /* Relative index since scene transition set to 0 */
  **end if**
  **if** $x <= K$ **then**
    $fcount\_at\_x[x] \leftarrow fcount\_at\_x[x] + 1$
    **if** $l_i^p = l_i^t$ **then**
      $lmatched\_at\_x[x] \leftarrow lmatched\_at\_x[x] + 1$
    **end if**
  **end if**
  $x \leftarrow x + 1$
**end for**
**return** $lmatched\_at\_x, fcount\_at\_x$

**ASiST@x Analysis**

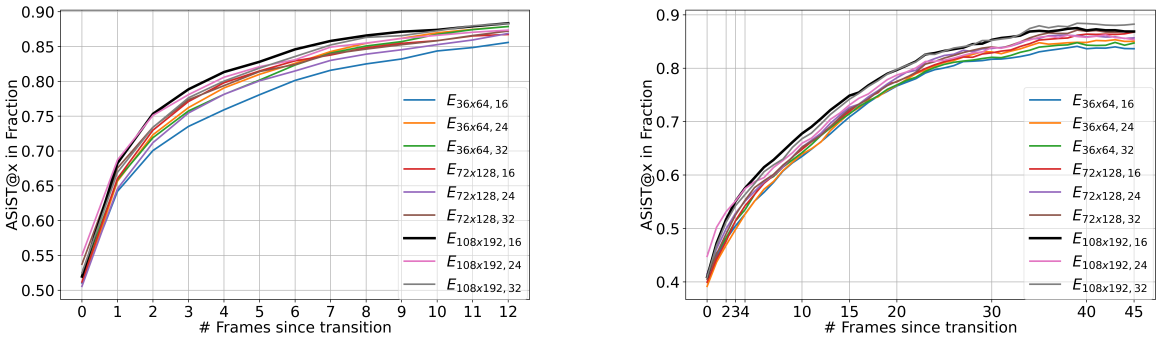
Due to the absence of per-frame accuracy metrics for both Layers of the HDD dataset, we are first to analyse them. To define the **ASiST@x** metric (**A**ccuracy at the $\mathbf{x}^{th}$ frame **Si**nce **S**cene **T**ransition), let us start by denoting the true label and predicted label of each frame $f_i$ as $l_i^t$ and $l_i^p$ respectively. Here, instead of calculating the *conventional* accuracy metric by comparing the aligned true and predicted label of each frame (which apart from being a high level abstract metric, is also not suitable for a heavily imbalanced dataset), we calculate it in the following way. In a sequence of true labels of a driving session, a scene transition occurs at the frame index $i$ (i.e. in frame $f_i$) when $l_i^t$ is not equal to $l_{i-1}^t$. Thus, a contiguous sequence of same valued true labels $l_m^t$ where $m \in [i, \cdots, j]$ denotes an ongoing driving scene in the frame sequence $F_{i,j}$ (i.e. the scene starts and ends at the frame index $i$ and $j$ respectively). Let $K$ denote the number of next frames since the index $i$ at which the scene has transitioned. Note that the set of $K$ frames does not include the frame $f_i$; therefore, after including $f_i$ in the set, the total number of frames in consideration for analysis increases to $K+1$ . Also, a scene can be of smaller duration than $K$ (next) frames, i.e. $j < i+K$, therefore, the actual number of scene frames into consideration is $n+1$ where $n = min(j-i, K)$. Next, let us define two zero-valued arrays: $lmatched\_at\_x = [0, \cdots, 0]$ and $fcount\_at\_x = [0, \cdots, 0]$ each of shape $K+1$. At index $x$, the $lmatched\_at\_x$ array stores the count of occurrences when $l_{i+x}^p = l_{i+x}^t$ and $fcount\_at\_x$ array stores the count of frames (since the scene transition) at index $i+x$ for $x \in [0, \cdots, n]$. Note that if $l_{i+x}^p = l_{i+x}^t$ at $x = 0$, then it implies that the driving scene was correctly recognized right at its apt transition. Also, if $l_{i+x}^p = l_{i+x}^t$ at $x > 0$, it is possible that the scene was recognized first at an earlier index in range $[i, \cdots, i+x-1]$. For a session, we define $\mathbf{ASiST@x} = \frac{lmatched\_at\_x}{fcount\_at\_x}$ (element wise division) after computing the $lmatched\_at\_x$ and $fcount\_at\_x$ for its scenes. Thus, the metric ASiST@x not only measures the efficacy of a model to recognize a scene at the $x^{th}$ frame after its true transition, but also implicitly measures the continuity of recognizing an occurring scene. In other words, ASiST@x scores tell us the percentage of scenes (that are at least $x+1$ frames long) that have been recognized by the model by the arrival of the $x^{th}$ frame since its transition.

In **Algorithm 1** we present an efficient implementation for calculating the ASiST@x metric. After obtaining the $lmatched\_at\_x$ and $fcount\_at\_x$ arrays for each of the test sessions in the HDD dataset, we calculate the overall ASiST@x $= \frac{\sum_{\text{session}} lmatched\_at\_x}{\sum_{\text{session}} fcount\_at\_x}$ (element wise summation, element wise division). Note that during ASiST@x scores calculation, the scene transition from an ongoing event (e.g. left turn) to the background class (i.e. no ongoing event) is also considered. It can be inferred from our definition of the ASiST@x

| **Elements governing the ASiST@x curve form** | | | **Effect on ASiST@x curve form** $x \in [0, \cdots, K]$ |
|---|---|---|---|
| **Scene duration compared to K + 1 frames** | $x^{th}$ **frame at which the scene is recognized first** | **Recognition continuity: Continuous or Irregular?** | |
| All $< K+1$ | All scenes recognized right at $x = 0$ | Continuous | ASiST@x = 100% $\forall$ $x$ up to a certain value, then the curve drops to 0 |
| __"__ | __"__ | Irregular after $x = k$ | ASiST@x = 100% $\forall$ $x \in [0, \cdots, k]$, then curve wiggles and drops to 0 |
| __"__ | Scenes recognized at $x \geq 0$, some not recognized at all | Continuous | ASiST@x < 100% at $x = 0$, then curve rises (may wiggle and reach 100% if individual duration vary), then drops to 0 |
| __"__ | __"__ | Irregular after $x = k$ | ASiST@x < 100% at $x = 0$ then curve rises (may wiggle and reach 100% if individual duration vary) then wiggles, and drops to 0 |
| Some (or None) $< K+1$, rest $\geq K+1$ | All scenes recognized right at x=0 | Continuous | ASiST@x = 100% $\forall$ $x$ |
| __"__ | __"__ | Irregular after $x = k$ | ASiST@x = 100% $\forall$ $x \in [0, \cdots, k]$, then curve wiggles |
| __"__ | Scenes recognized at $x \geq 0$, some not recognized at all | Continuous | ASiST@x < 100% at $x = 0$, then curve rises (may wiggle and reach 100% if few scenes' duration $< K+1$), and peaks and may plateau/wiggle/fall slightly |
| __"__ | __"__ | Irregular after $x = k$ | ASiST@x < 100% at $x = 0$, then curve rises up to $x = k$, but does not reach 100% if $k <$ all scenes' duration (otherwise may reach 100%), and then wiggles |

Table 4.5: Influence of the governing elements on ASiST@x curve form. Note: Above analysis is subject to variability due to the stochasticity of scenes duration and the value of $k$.

metric that it is governed by three elements, namely: (1) Duration of the scenes, (2) $x^{th}$ frame at which the scene transition is recognized, and (3) Recognition continuity of an occurring scene. The Table 4.5 details down the effect of these three governing elements on the curve form of the ASiST@x metric. It is apparent from the analysis in Table 4.5 that in general setting, the curves should ideally plateau at a high value as early as possible (with respect to $x$). Figure 4.2a and Figure 4.2b correspond to the ASiST@x scores for Layers 0 and 1 respectively. We set $K = 12$ for Layer 0 and $K = 45$ for Layer 1 since our frame sampling rate is set to 3 FPS and we found the mean duration of the scenes in the respective layers to be 3.82s (std: 2.77s) and 14.57s (std: 18.05s).




(a) HDD Layer 0: Goal-Oriented Action

(b) HDD Layer 1: Cause

Figure 4.2: **ASiST@x** plots for HDD dataset layers. Line corresponding to the best performing combination is in black. Best viewed in color.

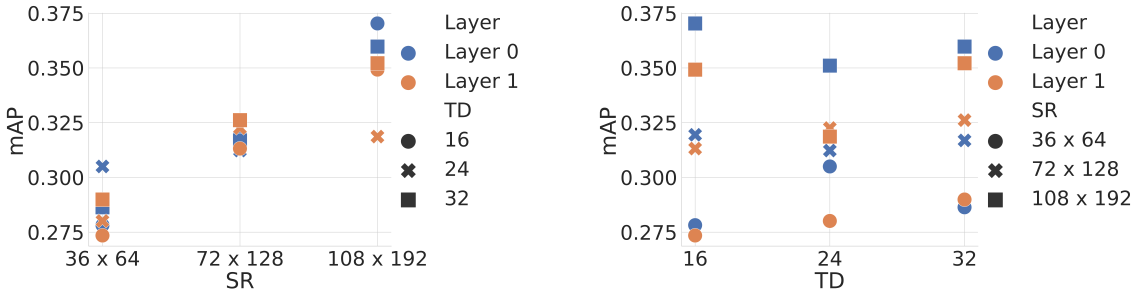
In Fig. 4.2a we see that at $x = 0$, $E_{108\times192,24}$ outperforms others by recognizing 55.02% of the Layer 0 driving scenes right at their true transition. At $x = 1$, this combination of spatial and temporal resolution again outperforms others by recognizing a scene transition 1 frame later (than its true transition) or 1 frame into its predicted transition (i.e. at the next frame if the scene transition was already recognized at $x = 0$) in 68.71% of the scenes ($E_{108\times192,16}$ performs nearly same - 68.24%). However, for $x >= 2$, $E_{108\times192,16}$ emerges as the clear winner. Considering its ASiST@3 score, our model takes only 1 second of time (since the true start of scenes) to correctly recognize a scene transition or an occurring scene (if the scene transition was recognized earlier) in 78.86% of all Layer 0 scenes. With respect to its ASiST@4 score, this combination of spatial and temporal resolution again enables our model to detect a scene transition at or earlier than 1.33 seconds (since true start) in 81.34% of all Layer 0 scenes which are at least 5 frames long (@ 3 FPS). We see that ASiST@x curves for all $E_{r\times c,l}$ keep rising, do not wiggle perceptibly, and do not fall within the range of $K$. This relates strongly to the analysis present in second last row of

the Table 4.5, thus hinting towards our model's ability ($\forall\ E_{r\times c,l}$) to continually recognize an occurring scene (after recognizing its transition earlier) apart from recognizing few extra scene transitions in the later frames. With respect to the scenes duration, we found that 35.41% of all scenes (in Layer 0 test data) had window size $< K + 1$ and 25.78% had window size $< 10$ frames.

In Fig. 4.2b, we see similar ASiST@x curve form for Layer 1 driving scenes, except that the curves are not as smooth as those in Fig. 4.2a. Considering the ASiST@0 score, we again see that $E_{108\times192,24}$ outperforms all others by achieving 44.78% score, but soon $E_{108\times192,16}$ takes over (at $x = 3$) and outperforms rest at higher values of $x$. From ASiST@3 score of $E_{108\times192,16}$ we see that it is able to recognize a scene transition at or earlier than 1s in 55.12% of all Layer 1 scenes which are at least 4 frames long (@ 3 FPS). We also see that $E_{108\times192,32}$ strongly contends with $E_{108\times192,16}$ at values of $x > 15$. This hints that increased temporal context might be necessary for recognizing longer duration scenes (recollect that mean duration of Layer 1 scenes is $14.57s$), but since the Layer 1 scenes' duration are highly variable (std: $18.05s$), it cannot be conclusively established. The ASiST@x curves for all $E_{r\times c,l}$ keep rising and seemingly plateau at values of $x$ closer to $K$ and perceptibly wiggle too. This wiggliness can be attributed to one or more of the following reasons: (1) it can be an image artefact due to the packing of comparatively (with respect to Figure 4.2a) large number of frames-since-transition, (2) it can be due to the highly variable duration of Layer 1 scenes - implying many scenes end earlier, (3) it can be due to the possible discontinuity in recognizing ongoing scenes. With respect to scenes duration, we found 44.95% and 33.72% of scenes (in Layer 1 test data) which were smaller than $K+1$ and 30 frames respectively.

### Effect Analysis of Spatial Resolution and Temporal Depth

Here we study the individual effects of spatial resolution and temporal depth variation on mAP while keeping the other constant. Figure 4.3a shows an increasing trend in the mAP scores for both Layers and different temporal depths as the spatial resolution increases. This suggests using higher spatial resolution inputs but it comes with an increased cost of computational requirements. Also, we found in Section 4.4.2 that a resolution of $108 \times 192$ pixels suffices the performance of $224 \times 224$ pixels resolution. Upon observing the class-wise AP scores, we found that increasing spatial resolution helps in better recognition of Left Lane Change, Right Lane Change, Left Lane Branch, Sign, Traffic Light, and Crossing Vehicle. In Figure 4.3b we do not see a definitive trend in the mAP scores as the temporal depth increases. Upon observing the class-wise AP scores for certain fixed spatial resolutions, we found that increased temporal depths resulted slightly better




(a) Varying Spatial Resolution (SR)

(b) Varying Temporal Depth (TD)

Figure 4.3: Individual effect of varying the Spatial Resolution and Temporal Depth. Layer 0 and Layer 1 are color coded. Best viewed in color.

performance for recognizing U-turn, Right Lane Change, Left Lane Change, and Crossing Vehicle. This subtly suggests that the determination of an ideal temporal depth is driving scene dependent, however this hypothesis requires further investigation. Since the mAP (in Figure 4.3b) does not strongly depend on the temporal depth, and the mAP scores with 16 frames temporal depth is higher or comparable to others, we favour the 16 frames temporal depth due to lesser computations. Conclusively, it can be said that a combination of $108 \times 192$ pixels spatial resolution and 16 frames temporal depth performs best for recognizing real time driving scenes in general setting.

## 4.5 Conclusion

From our extensively conducted experiments, we showed the success of our C3D inspired 3D-CNNs based model for Driving Scene Understanding. Our visuals only model was found to be comparable and outperformed a variety of visual and multi-modal Driving Scene Understanding approaches as seen in Tables 4.2, 4.3, and 4.4. In accordance with our intention to determining the scene transitions right at their onset, we introduced the ASiST@x metric to evaluate the efficacy of our approach. This metric can be extended to other datasets and different types of tasks as well which deal with the problem of continuous activity/scene recognition. We found that our model achieves ASiST@0 scores of 55.02% and 44.78% for Layers 0 and 1 of the HDD dataset respectively with an input resolution of $108 \times 192$ pixels and a temporal depth of 24 frames. In addition, our model takes just 1s of time (at 3 FPS) since the true start of scenes to correctly recognize scene transitions in 78.86% and 55.12% of Layer 0 and Layer 1 driving scenes respectively. We

experimentally found the combination of $108 \times 192$ pixels resolution and 16 frames temporal depth to be the best among other combinations for recognizing real time driving scenes in the largest real world public dataset. Owing to the demonstrated success of 3D-CNNs, we surmise that creation of multi-modal frameworks (to incorporate e.g. CAN Bus sensor data) with 3D-CNNs would push the results further. One can also explore increasing the Convolutional kernel size and developing shallower architectures to keep the number of trainable parameters in check, thereby leveraging higher spatial resolutions and examining the effect of shorter temporal depths. In addition, one may also segregate the HDD dataset scenes into groups with sufficiently varying means and low standard deviation (of scenes duration) to study the effect of temporal depths in detail, apart from developing more explicit metrics for detecting continuity in driving scene recognition. Finally, we hope that our insights in the necessary spatial resolution and temporal depth serve as the initial considerations when researchers toil over choosing these hyper-parameters.

# Acknowledgements

We would like to thank Yi-Ting Chen [89] for promptly helping us navigate through the HDD dataset. This work is supported by the NSERC Grant Award AWD-013766.

# Chapter 5

# Foreword to Article 2 in Chapter 6

Article 2 in Chapter 6 presents two neuromorphic hardware-friendly methods for implementing MaxPooling in SNNs. Such methods can be used to assist in the ANN-to-SNN conversion of the 3D-CNNs based model with 3D-MaxPooling layers (in Chapter 4) to an SNN which is entirely deployable on a neuromorphic hardware.

## Foreword

Spiking Neural Networks (SNNs) are an emerging domain of biologically inspired neural networks that have shown promise for low-power AI. A number of methods exist for building deep SNNs, with Artificial Neural Network (ANN)-to-SNN conversion being highly successful. MaxPooling layers in Convolutional Neural Networks (CNNs) are an integral component to downsample the intermediate feature maps, but the absence of their hardware-friendly spiking equivalents limits such CNNs' conversion to deep SNNs. In this chapter, we present two hardware-friendly methods to implement MaxPooling in deep SNNs, thus facilitating easy conversion of CNNs with MaxPooling layers to SNNs. In a first, we also execute SNNs with spiking-MaxPooling layers on Intel's Loihi neuromorphic hardware (with MNIST, FMNIST, & CIFAR10 dataset); thus, showing the feasibility of our approach. As mentioned in the previous paragraph, such spiking-MaxPooling approaches can also be used to implement 3D-MaxPooling in 3D-CNNs based SNNs. We note that SNNs with AveragePooling layers are easily (and entirely) deployable on neuromorphic hardware. However, the same 3D-CNNs based model, but with AveragePooling layers (which can be converted to AveragePooling based SNNs) performed poorly than MaxPooling based 3D-CNNs (more in Chapter 7); hence the need of spiking-MaxPooling.

# Chapter 6

# Spiking Approximations of MaxPooling in SNNs

## 6.1 Introduction

Artificial Neural Networks (ANNs) have established themselves as the de-facto tool for a variety of Artificial Intelligence (AI) tasks. And the flagship performance of CNNs for image recognition/classification has remained unparalleled so far. However, their limitations in the aspects of energy consumption, robustness against noisy inputs, etc. has attracted interest in developing their spiking counterparts. Spiking Neural Networks (SNNs) offer a promise of low power AI and have shown to be more robust against noisy inputs [105], perturbations to the weights [60], and adversarial attacks [99, 100]. Out of a number of ways to build SNNs [87] (section 2.2), the ANN-to-SNN conversion method has been highly effective for building deep SNNs. In this method, one first trains an ANN with traditional rate neurons (e.g. `ReLU`) and then replaces those rate neurons with spiking neurons, along with the other required modifications of weights [98], etc. For our work, we consider this ANN-to-SNN conversion paradigm to build deep SNNs.

MaxPooling in CNNs is a common method to downsample the intermediate feature maps obtained from Convolutional layers. One can also use AveragePooling or Strided Convolution to downsample the feature maps; however, the choice of the pooling method is contextual [15], and MaxPooling is generally found to give better performance. That said, [15] also found that "depending on the data and features, either max or average pooling may perform best". Several architectures e.g. ResNet-50 (-152) [36], Xception [20], EfficientNet [107] which form the backbone of different methods to achieve SoTA

results on ImageNet, use a mix of Max and AveragePooling layers (or GlobalMax/Average Pooling layers).

However, the conversion of CNNs with MaxPooling layers to SNNs is a convoluted process. MaxPooling in SNNs has been a long-standing problem and only a handful of approaches exist for the same. [41] present three approaches for MaxPooling in SNNs; where they design a pooling gate to monitor the spiking neurons' activities (in a pooling window) and dynamically connect one of the neurons to the output neuron based on their criteria for the maximally firing neuron. Such a gating mechanism is leveraged by [94] too, where they employ a finite impulse response filter to control the gating function and do MaxPooling in SNNs; [53] too use the same. Few other works [69, 118, 82, 63, 75] leverage a Time-To-First-Spike based temporal Winner Take All (WTA) mechanism (or its variants) to do MaxPooling in SNNs; where the earliest occurring spike in a pooling area is sent to the next layer, with rest of the neurons (in the pooling area) reset to blocked. Similarly, [65] employ a lateral inhibition based method to do MaxPooling in SNNs. In another novel approach, [80] select a neuron (in the pooling region) with the highest membrane potential to output the spikes; they do a soft reset (i.e. reset by subtraction) of the firing neuron's membrane potential. They also propose a hardware architecture for their method of MaxPooling. However, none of the above methods have been evaluated on a specialised neuromorphic hardware (e.g. Loihi [21]), with the exception of WTA (but on FPGA [82]).

The conversion of CNNs with AveragePooling layers is trivial, as the AveragePooling layers can be modeled as convolution operation in the SNNs. Thus, a number of works replace the MaxPooling layers with AveragePooling layers [115, 98, 18, 31, 56, 57, 117, 43] or with Strided Convolutional layers [28, 85] in their network; often leading to weaker ANNs [94]. Overall, this dearth of spiking-MaxPooling approaches in SNNs and the neuromorphic hardware-unfriendliness of the existing ones motivated us to build spiking equivalents of the MaxPooling operation which can be entirely deployed on a neuromorphic hardware (e.g. Loihi). Our contributions are outlined below:

- We propose two methods to do MaxPooling in SNNs, and evaluate them on the Loihi neuromorphic chip
- In a first, we also deploy deep SNNs with MaxPooling layers on the Loihi boards with one of our methods

We next lay out our **Methods** of spiking-MaxPooling, followed by the **Experiments & Results** section, and a **Discussion** on the result analysis and adaptability of our methods.

## 6.2 Methods

In this section we present our proposed methods of spiking-MaxPooling. We start with each method's elemental details, followed by their Proof-of-Concept Demonstration on the Loihi chip. Henceforth, otherwise stated, all the instances of "neurons" are Integrate & Fire (IF) spiking neurons.

### 6.2.1 Method 1: MAX join-Op

MAX join-Op (MJOP) based spiking-MaxPooling leverages the low level NxCore APIs made available through the NxSDK tool (to program the Loihi chips) by Intel; thus, this method is Loihi hardware dependent. A single Loihi chip consists of 128 Neuro-Cores, where each Neuro-Core implements 1024 Single Compartment (SC) spiking units. Each compartment can be simulated as an individual neuron or can become part of a Multi-Compartment (MC) neuron. Each MC neuron is a binary tree of single compartments, where each node (a compartment) can have at most two child nodes/dendrites (also compartments). Note that a MC neuron cannot span across two or more Neuro-Cores; rather is limited to just one Neuro-Core. Thus, a MC neuron can have a maximum of 1024 SC. Through each connection between the compartments (in a MC neuron), the two state variables: current ($U$) and voltage ($V$) can flow. Thus, a compartment can communicate either its $U$ or $V$ to its parent compartment. The parent compartment incorporates the state variables from its dendrites with its own $U$ by following a *join* operation defined for its dendritic connections. Note that, an external $U$ can be injected to each compartment (including the root compartment) in a MC neuron, thereby enabling them to spike (provided their $V$ reaches the threshold). Also, note that the neurons (MC or SC) in a Loihi chip communicate via spikes only; they cannot exchange $U$ or $V$ directly, which falls in line with the biological neurons.

Fig. 6.1 shows an example of a MC neuron communicating its spikes to a SC neuron. The MC neuron has a binary tree structure, with the root node (i.e. soma compartment `C`) having two child leaf nodes (dendrite compartments `A` and `B`). In Loihi, each compartment's voltage i.e. $X.V[t]$ ($X$ can be a soma or its dendrites) is updated by the following rule:

$$X.V[t] = X.V[t-1] \times (1 - decay) + X.dV[t] \tag{6.1}$$

where the value of $X.dV[t]$ is found by applying the specific *join* operation (join-Op) e.g. ADD, MAX, MIN, etc. over its current $X.U[t]$ and the state variables from its dendrites. We leverage this MC neuron creation functionality and the MAX join-Op for realizing

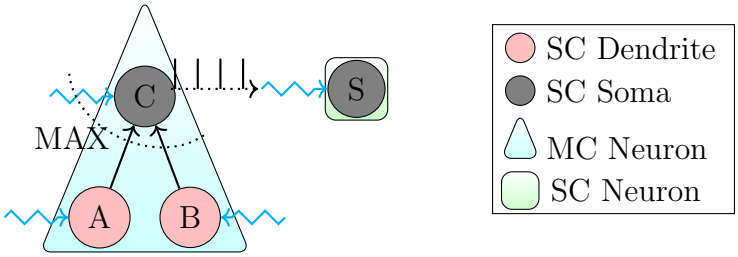



Figure 6.1: SC: Single Compartment, MC: Multi-Compartment. Wiggly arrows denote post-synaptic current due to the incoming spikes; Dotted straight arrow shows the resulting spike-train due to `C`'s spiking activity; Dotted curved line over 3 arrows shows the 3 join-Op arguments.

spiking-MaxPooling (on Loihi boards) in deep SNNs. We next explain the MAX join-Op in the context of the MC neuron in Fig. 6.1. We begin by defining a few terms first: $X.U[t]$ and $X.bias$ denote the input current and bias current (respectively) of a compartment $X$. $X.U'[t]$ is the sum of $X.U[t]$ and $X.bias$. $A[t]$ (and $B[t]$) is the output of the child dendrite `A` (and `B`), which can either be $A.V[t]$ or $A.U'[t]$ (and $B.V[t]$ or $B.U'[t]$) depending upon which state variable we want to work with; we choose $U$. NxSDK defines the MAX join-Op as ($X$ being `C` here):

$$C.dV[t] = max(C.U'[t], A[t], B[t]) \tag{6.2}$$

We expand and simplify the Eq. 6.2 by assuming $X.bias = 0$ for $X \in \{\texttt{C}, \texttt{A}, \texttt{B}\}$ and setting the child compartments `A` and `B` to output current instead of $V$ (to its parent `C`). Thus, $A[t] = A.U'[t]$ and $B[t] = B.U'[t]$. Eq. 6.2 simplifies as:

$$\begin{aligned} C.dV[t] &= max(C.U'[t], A[t], B[t]) && (6.3)\\ &= max(C.U[t] + C.bias, A[t], B[t]) && (6.4)\\ &= max(C.U[t], A.U'[t], B.U'[t]) && (6.5)\\ &= max(C.U[t], A.U[t], B.U[t]) && (6.6) \end{aligned}$$

Thus, in MJOP case, the root compartment (i.e. soma) `C`'s voltage dynamics is governed by the maximum of the input currents to it (from external source and its dendrites `A` & `B`).

### MJOP Net for MaxPooling:

In a conventional CNN architecture, MaxPooling is done over the activations of the preceding Convolutional layer rate-neurons. In an SNN, we can represent those real-valued

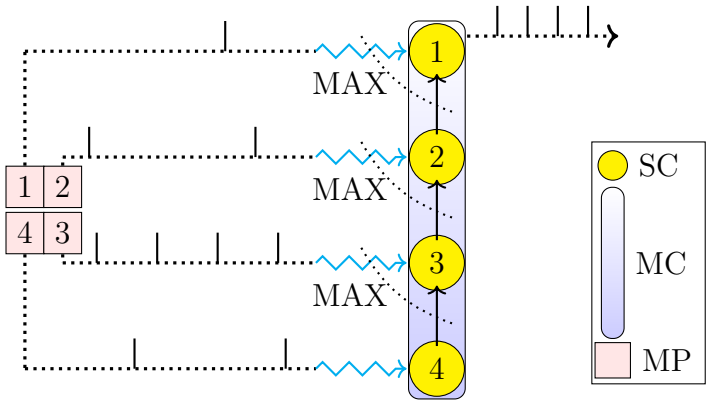



Figure 6.2: *MJOP Net.* SC: Single Compartment, MC: Multi-Compartment Neuron, MP: $2 \times 2$ MaxPooling Window. Integers $(1, 2, 3, 4)$ simply show a one-to-one correspondence.

activations (of rate-neurons) by passing the corresponding spiking-neurons' spike-trains through a low-pass filter (also known as filtering/synapsing). In other words, the synapsed spikes i.e. the post-synaptic current $U$ represents the activation. We leverage this characteristic of SNNs and feed the individual currents $U_i$ (in a pooling window) to a MAX join-Op configured MC neuron. Note that the number of compartments in the MC neuron should be equal to the size of the pooling window.

For a $2 \times 2$ MaxPooling window, we construct a MC neuron with 4 compartments as shown in the Fig. 6.2. The outgoing spikes from each neuron in a pooling window induce a post-synaptic current $U$ in the respective individual compartments (*bias* current of each compartment is set to 0). Each of the compartments (except the root/soma) is set to communicate its $U$ to its parent. The leaf node/compartment 4 upon receiving the post-synaptic current updates its $V$ and communicates the received $U$ to its parent 3. Compartment 3 then computes the MAX of the current from its child 4 and the incoming post-synaptic current, updates its $V$, and communicates the resulting maximum $U$ to its parent 2. Note that in this network, MAX join-Op is executed over two arguments. Compartments 2 and 1 repeat this same process; except that the compartment 1 has no parent to communicate the MAX $U$ so far. The soma compartment 1 then spikes at a rate *corresponding* to the maximum input post-synaptic $U$ depending on its configuration, instead of outputting max $U$.

**Proof-of-Concept Demonstration:**

In MJOP Net, a running MAX of input currents $U_i$ is maintained, which is finally fed to the root/soma compartment. Mathematically:

$$U_{out} = F(G(max(U_1, max(U_2, max(U_3, U_4)))))) \tag{6.7}$$

where $U_i$ is the input current to compartment $i$, $G$ is the non-linear dynamical function of $U$ governing the voltage dynamics (thus, the spiking output) of soma, $F$ is the synaptic filter applied on the spike outputs of soma. Since the soma does not communicate the computed max current, rather its spikes to the next neuron (if connected), there arises a need to properly *scale* the synapsed spikes i.e. $U_{out}$ to match "True Max $U$" ($= max(U_1, U_2, U_3, U_4)$ computed without spiking neurons on a non-neuromorphic hardware). We next execute the MJOP Net (in Fig. 6.2) on the Loihi chip. The net consists of a MAX join-Op configured MC neuron of 4 compartments receiving periodic spiking inputs (spike amplitude 1, with time-periods of 10, 8, 4, and 6 - a possible case when receiving the spike outputs of pooled neurons in a preceding Convolutional layer in an SNN). We set a probe on soma (compartment 1) and filter its output spikes. In Fig. 6.4a we see that the "Scaled $U_{out}$" matches the "True Max $U$" closely; the "Average $U$" is lower than the estimated max.

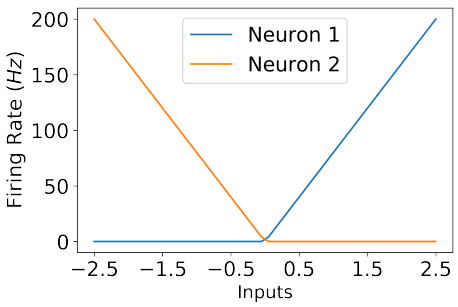
### 6.2.2 Method 2: Absolute Value based Associative Max

The representation of the real-valued activations of the rate-neurons as currents in SNNs inspires our second method as well. The Absolute Value based Associative Max (AVAM) method of spiking-MaxPooling is hardware independent and leverages the following two properties of the $max$ function.

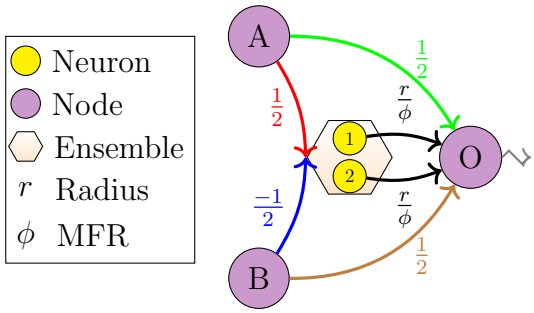
$$max(a, b) = \frac{a+b}{2} + \frac{|a-b|}{2} \tag{6.8}$$

$$\begin{aligned} max(a_1, a_2, a_3, \cdots, a_n) = max(&max(a_1, a_2), \\ &\cdots, max(a_{n-1}, a_n)) \end{aligned} \tag{6.9}$$

where $a_i, a, b \in \mathbb{R}$ and $n \in \mathbb{N}$; Eq. 6.9 holds due to the Associative Property of $max()$. In Eq. 6.8, the average term $\frac{a+b}{2}$ (a linear operation) can be easily calculated by the connection weights from the neurons representing those values (i.e. $a$ and $b$); the challenge is to calculate the non-linear absolute value function i.e. $|.|$. One can use the Neural Engineering Framework (NEF) principles [26] to estimate the $|.|$ function by employing




(a) Tuning curves of two neurons



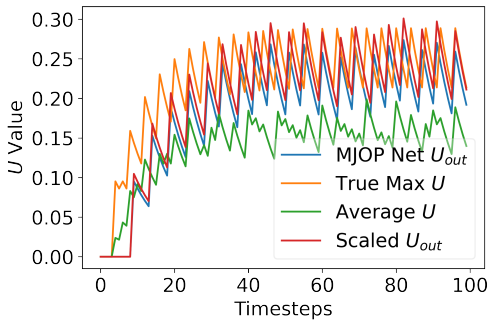

(b) $max(a, b)$ Network

Figure 6.3: MFR: Max Firing Rate; Node: A programming construct to either represent a value or sum the inputs, and forward the same.

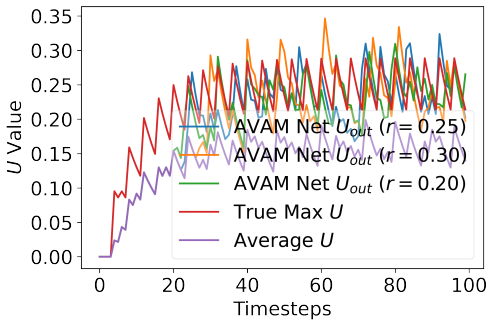
a network of *Ensemble*s of neurons. However, the number of neurons required in each *Ensemble* can be large (100$s$ or more); thus, this method is not scalable. For the same reasons, a direct calculation of the $max()$ using NEF principles is not desirable.

**Estimation of |.| function:**

Therefore, we rather take a unique approach of using the tuning curves to estimate the |.| function. Tuning curves characterize the activation (i.e. firing rate) profile of neurons with respect to the input stimulus. In NEF, these curves depend on the neuron type and properties (e.g. max firing rate $\phi$, representational radius $r$, etc.). Upon configuring an *Ensemble* of two IF neurons properly (e.g. $\phi = 200$Hz, $r = 2.5$), one can obtain desirable tuning curves as shown in the Fig. 6.3a, which resembles the plot of the |.| function; we leverage the same to estimate the absolute value of a signed scalar input. In our example (Fig. 6.3a), "Neuron 1" (and "Neuron 2") is tuned to fire at $\phi = 200$Hz when 2.5 (and $-2.5$) is fed to the *Ensemble*. Thus, irrespective of the sign of the input values $\in [-r, \cdots, r]$, we receive a positive firing rate from either neuron. Therefore, for an input $r$ (or $-r$), we can filter the spiking output from the corresponding neuron (the other outputs 0) to obtain $\phi$ and scale it with $\frac{r}{\phi}$ (i.e. $\phi \times \frac{r}{\phi}$) to estimate $|r|$. Note that for such a system of two neurons, one needs to pre-determine a max firing rate $\phi$ and the approximate representational radius $r$. If the magnitude of the input value is lesser (or larger) than $r$, then this system produces a noisier (and saturated) output.




(a) MJOP Net $U_{out}$




(b) AVAM Net $U_{out}$

Figure 6.4: "MJOP Net $U_{out}$" is the synapsed/filtered spiking output from the soma. It is scaled by 1.1 (to obtain "Scaled $U_{out}$") to match the "True Max $U$" $= max(U_1, U_2, U_3, U_4)$ closely. "AVAM Net $U_{out}$" are the outputs from Node `O` (in Fig. 6.5) for different radii $r$. For $r = 0.25$ and 0.20, the $U_{out}$ matches the "True Max $U$" $= max(U_1, U_2, U_3, U_4)$ closely.

**Estimation of $max(a, b)$:**

The above method of estimating the $|.|$ function can be incorporated with the average term calculation to construct a network as shown in the Fig. 6.3b to estimate the $max(a, b)$. In Fig. 6.3b, Nodes `A` and `B` represent and output the values $a$ and $b$ respectively as currents. They are directly connected to the Node `O` with a connection weight of $1/2$ each. Thus, their scaled output i.e. $a/2$ and $b/2$ gets summed up at Node `O` to result in $(a + b)/2$. Nodes `A` and `B` are also connected to an *Ensemble* of two Neurons, `1` and `2` (whose tuning curves are similar to that in Fig. 6.3a and their representational radius $r \approx (|a - b|)/2)$ with a connection weight of $1/2$ and $-1/2$ respectively, such that the sum $(a - b)/2$ fed to the *Ensemble*. Note that $r$ can be heuristically set without knowing the values of $a$ and $b$ (shown later). Depending on the sign of the sum $(a - b)/2$, either the Neuron `1` or Neuron `2` spikes at a frequency $\phi$ (the other outputs 0). Therefore, after filtering the spike outputs to obtain $\phi$ and scaling it with $\frac{r}{\phi}$ $(= \phi \times \frac{r}{\phi})$ through the weighted connection to the Node `O`, $r$ is sent. The Node `O` then finally accumulates the inputs, i.e. $\frac{a+b}{2} + r \approx \frac{a+b}{2} + \frac{|a-b|}{2}$ which is approximately equal to the $max(a, b)$ and relays the same.

**Proof-of-Concept Demonstration:**

For a 2×2 MaxPooling window, we can compute the $max(a_1, a_2, a_3, a_4)$ as $max(max(a_1, a_2), max(a_3, a_4))$ using the Associative Property of $max()$. We therefore construct the AVAM

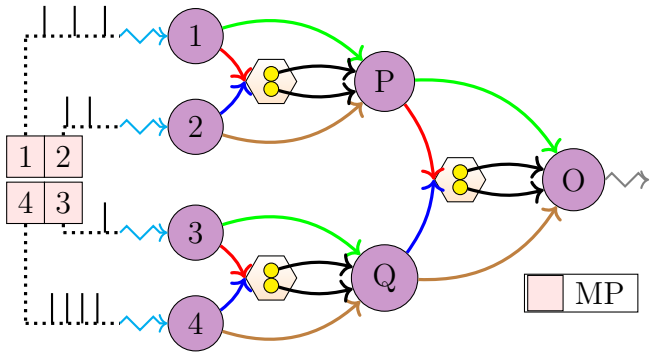



Figure 6.5: *AVAM Net for* $2 \times 2$ *MaxPooling.* MP: $2 \times 2$ MaxPooling Window. Edges are color coded; refer Fig. 6.3b legend.

Net, shown in Fig. 6.5 with four spiking inputs (firing time-period $= 10, 8, 4, 6$) connected to the individual Nodes. Note that the filtered spikes i.e. currents $U_i$ are being fed to the Nodes (here $a_i = U_i$) and they relay the same to the next connected components. In this hierarchical network, $max(a_1, a_2)$ and $max(a_3, a_4)$ gets estimated at the Nodes `P` and `Q` respectively. They forward the same and Node `O` finally estimates the $max(max(a_1, a_2), max(a_3, a_4))$. Three instances of this AVAM Net are executed on the Loihi chip with IF neurons' $\phi$ fixed to 500Hz and $r \in \{0.20, 0.25, 0.30\}$. In each instance, all the neurons in each *Ensemble* had the same $\phi$ and $r$. Each instance's estimated $max(U_1, U_2, U_3, U_4)$ i.e. "AVAM Net $U_{out}$" (for a corresponding $r$) is shown in the Fig. 6.4b. We see that $r$'s value around 0.25 (for a fixed $\phi$) does a fair job of approximating the "True Max U"; as well as, the estimated max $U$ are higher than the Average $U$.

One should note that ideally, the output current from a MaxPooling window (of spiking neurons) should be the current due to the maximally firing neuron (i.e. "Ideal Max $U$"). However, the MJOP and AVAM spiking-MaxPooling methods compute the instantaneous max of all incoming currents (in a pooling window) at each time-step, which is not equivalent to the "Ideal Max $U$". It is possible that the instantaneous max of currents could be higher than the ideal max when a slower spiking neuron fires recently than the maximally firing neuron. This holds true with "True Max $U$" as well; however, we defined it as $max()$ of currents in the spirit of MaxPooling in ANNs. Therefore, our methods are approximations of the ideal MaxPooling in SNNs.

### 6.2.3 Heuristics for *scale* (MJOP) and *radius* (AVAM)

As seen in the **Proof-of-Concept Demonstration** sections, one needs to properly *scale* the $U_{out}$ in case of MJOP Net and set the *radius* parameter in AVAM Net to correctly approximate the "True Max $U$". For a fixed configuration of compartments/neurons in the MJOP and AVAM Net, the value of *scale* and *radius* depends on the group of inputs $U_i$. Recollect that in the MJOP Net, the soma compartment spikes at a rate *corresponding* to the maximum input $U_i$, and the output $U_{out}$ needs to be scaled accordingly. And in the AVAM Net, the *radius* should be heuristically chosen to be equal to $|a-b|/2$ for estimating the $max(a, b)$ (where $a$ and $b$ are the inputs $U_i$). The inputs $U_i$ in turn depend on the periodicity (or the Inter-Spike Interval (ISI)) of the incoming individual spike trains. We note here that the maximum and minimum value of $U_i$ can be 1 (with spike amplitude $= 1$, ISI $= 1$) and 0 (with the corresponding neuron not spiking at all) respectively. This implies that the maximum value of the difference of two $U_i$s can be 1, i.e. $radius \leq 1$ always. This holds true for the *radius* value of the *Ensemble* neurons further in the AVAM Net hierarchy. Moreover, many or all of the neurons in a pooling window may spike with ISI $> 1$, which would further lower down the *radius* value.

One can heuristically choose the *scale* and *radius* values by analysing the ISI distribution of the neurons in a model's Conv layer (preceding the MaxPooling layer) for a dataset; although, these values may be required to be tuned further. We therefore conduct a toy experiment where we collect the ISIs of 2048 neurons in a Conv layer (of one of our converted SNNs) for 1000 training images of MNIST and CIFAR10 each, and construct the respective ISI distributions (averaged across 1000 images). From each distribution, we obtain 256 groups of randomly sampled ISIs (group size 4) and filter the respective spiking inputs to create 256 groups of $\{U_1, U_2, U_3, U_4\}$. We next compute the "True Max $U$" and "Estimated Max $U$" for different values of *scale* and *radius* in the MJOP & AVAM Net respectively, and analyse their effects (refer Fig. 6.6). Note that the "Average" $U$ values (synonymous to AveragePooling) are $\leq$ the respective estimated max $U$. The high correlation of the true and estimated $U$ obtained for a varied group of spiking inputs shows the efficacy of our spiking-MaxPooling methods. For our MJOP & AVAM Net, *scale* value around 2.0 and a *radius* value around 0.3 fairly approximates the "True Max $U$". Note that in the AVAM Net, ideally, the *radius* of each *Ensemble*'s neurons should be set independently to estimate the varying $|a - b|/2$, however for simplicity, we keep it same.

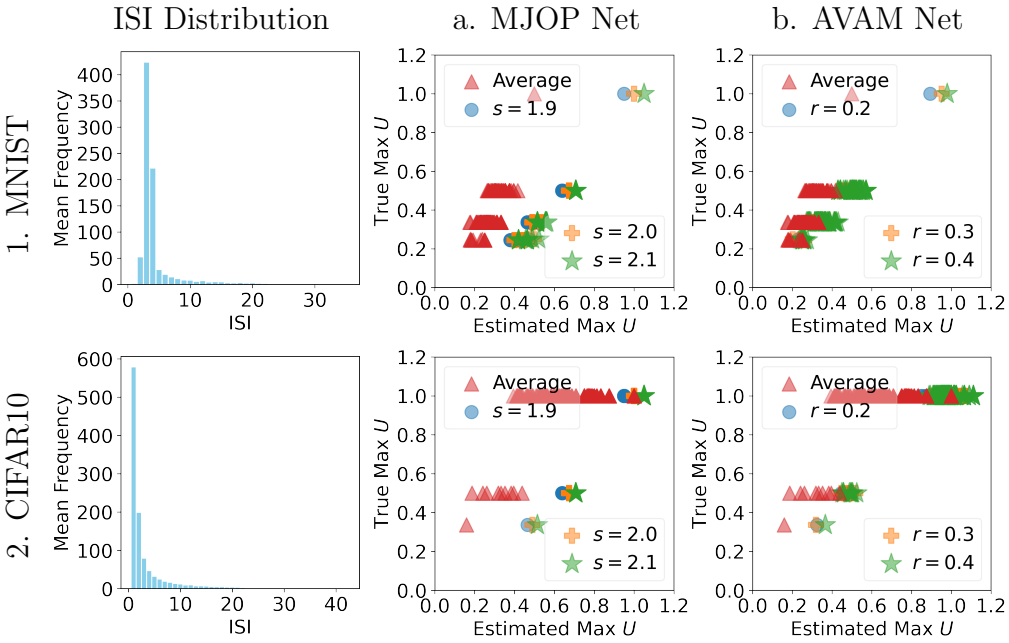



Figure 6.6: *Scatter-Plots of MJOP & AVAM Net Outputs* over spiking inputs (their ISI's in groups of 4). $s$: Scale, $r$: Radius. Correlation Coefficients: (1, a) 0.98; (1, b) 0.99; (2, a) 0.99, (2, b) 0.95.

## 6.3 Experiments & Results

We next describe the experiments conducted with the MJOP and AVAM methods of spiking-MaxPooling. In accordance with the ANN-to-SNN conversion paradigm, we first train a rate-neuron (`ReLU`) based model and then convert it to a spiking network (of IF spiking neurons). We use the NengoDL library [91] for training, conversion, and inference; and the NengoLoihi library for deploying the SNNs on the Loihi boards. While training the models we ensure that they are properly tuned to account for the firing-rate quantization of IF spiking neurons. In the converted SNNs, we do the MaxPooling operation via our proposed methods of spiking-MaxPooling Nets. We use MNIST, FMNIST, and CIFAR10 datasets (normalized $[-1, 1]$) and conduct experiments with 3 different architectures (Fig. 6.7). For simplicity, we fix the MaxPooling window to $2 \times 2$ in all architectures.

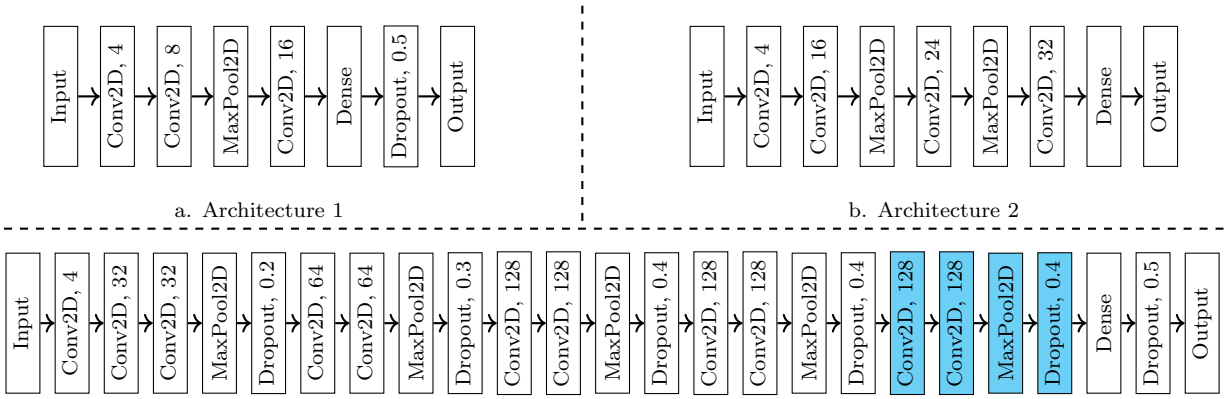



Figure 6.7: *CNN Architectures.* "Conv2D, x" $\Rightarrow$ "x" number of filters in the Conv layer; "Dropout, x" $\Rightarrow$ "x" dropout probability. In each architecture, Conv layer strides $= (1, 1)$, kernel size $= (3, 3)$ (except the first Conv layer with kernel size $= (1, 1)$); $2 \times 2$ MaxPooling window; 128 neurons in Dense layer; no activation in the Output layer; no *bias* in all layers except Dense. MaxPool layers in all architectures have no padding; Conv layers too have no padding except in Architecture 3. For experiments with MNIST & FMNIST, the colored blocks in Architecture 3 are removed to account for their smaller image size than CIFAR10.

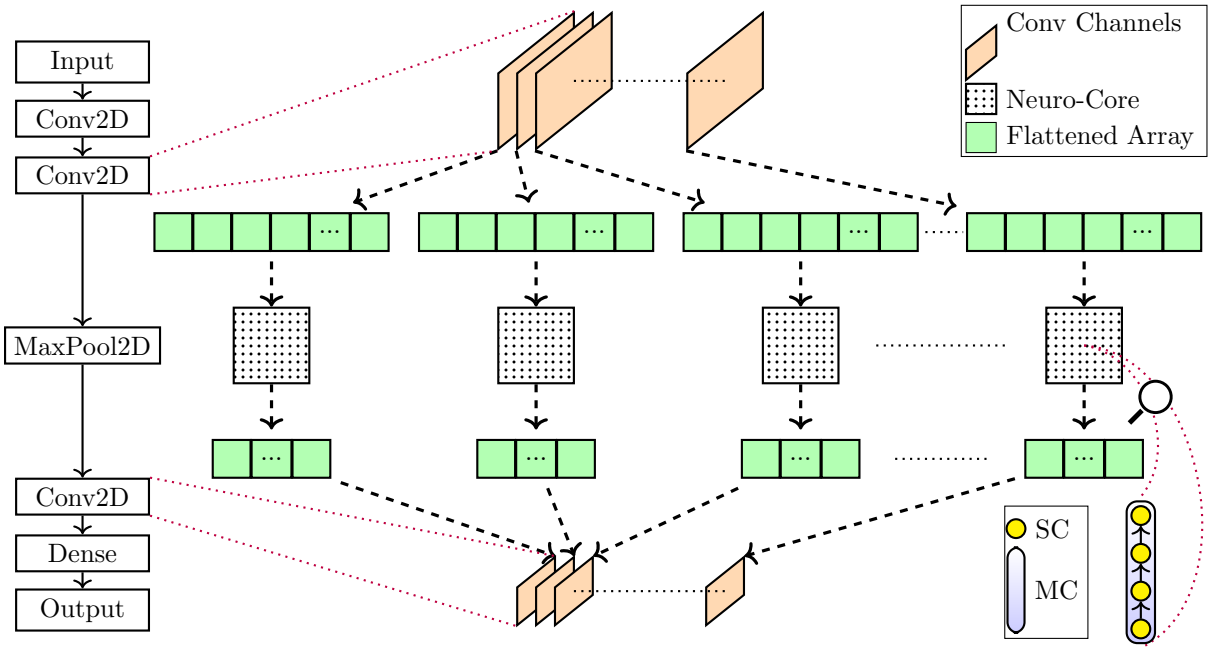



Figure 6.8: *MAX join-Op based MaxPooling.* For a MaxPool layer in an architecture, the preceding Conv layer's channels are each flattened and mapped to a single Neuro-Core. Each Neuro-Core has an *Ensemble* of MJOP Net configured MC neurons. Post execution of MJOP Nets on Neuro-Cores, the flattened vectors are reshaped to channels and passed to the next Conv layer.

| | Architecture 1 | | | | | | | | | | Architecture 2 | | | | | | | | | Architecture 3 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NSR | TMS | MAS | MJOP | | | AVAM | | | | NSR | TMS | MAS | MJOP | | | AVAM | | | | NSR | TMS | MAS | AVAM | | | |
| | | | | $S_a$ | $S_b$ | $S_c$ | $R_a$ | $R_b$ | $R_c$ | $R_d$ | | | | $S_a$ | $S_b$ | $S_c$ | $R_a$ | $R_b$ | $R_c$ | $R_d$ | | | | $R_a$ | $R_b$ | $R_c$ | $R_d$ |
| **CIFAR10** | 60.2 | 60.6 | 48.7 | 55.0 | 55.1 | 51.8 | 60.4 | 60.5 | 60.7 | 59.5 | 65.3 | 64.9 | 38.8 | 55.7 | 51.8 | 26.3 | 64.1 | 64.6 | 65.0 | 64.2 | 83.7 | 82.8 | 69.6 | 82.7 | 82.7 | 82.7 | 81.3 |
| **MNIST** | 98.8 | 98.8 | 98.2 | 98.2 | 98.2 | 98.2 | 98.7 | 98.7 | 98.7 | 98.6 | 99.1 | 98.7 | 98.3 | 97.9 | 98.0 | 97.2 | 98.8 | 98.8 | 98.8 | 98.9 | 99.3 | 99.4 | 98.9 | 99.3 | 99.4 | 99.4 | 99.4 |
| **FMNIST** | 91.0 | 90.5 | 87.3 | 88.0 | 88.1 | 88.0 | 90.2 | 90.2 | 90.3 | 89.4 | 89.9 | 89.5 | 79.5 | 86.4 | 86.3 | 85.8 | 89.0 | 89.2 | 89.1 | 88.3 | 93.4 | 93.2 | 91.6 | 93.2 | 93.3 | 93.4 | 93.2 |

Table 6.1: *Accuracy Results (%).* **NSR**: **N**on-**S**piking **R**eLU results; **TMS**: **T**rue-**M**ax $U$ **S**NN's results; **MAS**: **M**ax-to-**A**vg **S**NN's results; **MJOP** & **AVAM**: Spiking-MaxPooling SNN's results. For **CIFAR10** $S_a, S_b, S_c$ $(scale)$ are $1.0, 2.0, 5.0$; for **MNIST** they are $1.0, 1.2, 2.0$; for **FMNIST** they are $1.0, 1.5, 2.0$ resp. $R_a, R_b, R_c, R_d$ $(radius)$ $\forall$ datasets are $0.20, 0.25, 0.30, 1.0$ resp. As expected, $R_d = 1.0$ results in accuracy loss, since difference of $U_i$ is not always $\approx 1.0$; thus, poor approximation of max $U_i$.

### 6.3.1 Common settings in MJOP and AVAM methods

For our proposed spiking-MaxPooling methods, each Conv-channel is flattened and the $2 \times 2$ pooling window inputs are grouped in sizes of 4, and passed next to the layer of MJOP Nets or AVAM Nets depending on the choice of spiking-MaxPooling. The outputs from the individual Nets are collected in a channel wise manner and passed to the next Conv layer. While flattening the channels and passing the pooled inputs to the spiking-MaxPooling layer, and subsequently while collecting the outputs, one has to properly arrange the inputs and outputs to preserve the $2 \times 2$ MaxPooling layout.

### 6.3.2 Model Details

The first Conv layer in each architecture (in Fig. 6.7) has a kernel size of $(1, 1)$ which acts a pixel-value to spike converter. For training each architecture, we use the `Adam` optimizer with a learning rate of $1e{-}3$ and a decay of $1e{-}4$, and use the `Categorical Crossentropy` loss function with logits. For MNIST, Architectures 1, 2, and 3 were trained for 8 epochs each; for CIFAR10, 1 and 2 were trained for 64 epochs each, and 3 for 164 epochs; and for FMNIST, 1 and 2 were trained for 24 epochs each, and 3 for 64 epochs. The training was done on the NVIDIA Tesla P100 GPUs. During inference with SNNs, the individual test images (irrespective of the dataset) were presented for 50, 60, and 120 time-steps to the Architectures 1, 2, and 3 respectively.

### 6.3.3 SNNs with MJOP Net based spiking-MaxPooling

Since MJOP method is Loihi dependent (and not supported on the GPUs), we execute the converted SNNs straightaway on the Loihi boards in inference mode. Fig. 6.8 shows an example of how MJOP spiking-MaxPooling can be done in any arbitrary architecture. The individual channels of the preceding Conv layer are flattened and mapped to a Neuro-Core each. Each Neuro-Core has an *Ensemble* of MAX join-Op configured MC neurons (with 4 compartments as shown in Fig. 6.2) to do the $2 \times 2$ MaxPooling. The outputs (quartered in length) from each *Ensemble* deployed on individual Neuro-Cores are collected and reshaped as individual channels, and fed to the next Conv layer. This process is repeated for any additional MaxPooling layers in the network. While deploying the architectures on the Loihi boards, the first Conv layer and the last "Output" layer are executed Off-Chip; rest of the layers run On-Chip. Inference is done over all the test images in each dataset for varying *scale* values. Due to the Loihi hardware resource constraints and no support for `same` padding (in Conv layers) in NengoLoihi (1.1.0.dev0), we were unable to execute Architecture 3 on the Loihi boards. Table 6.1 shows the accuracy results for Architectures 1 and 2 (both executed on Loihi boards).

### 6.3.4 SNNs with AVAM Net based spiking-MaxPooling

AVAM method of spiking-MaxPooling is hardware independent. Although the individual AVAM Nets with varying radii were executed on Loihi (Fig. 6.4b), executing SNNs with them on the Loihi boards gets challenging due to the AVAM Net layers exceeding the maximum supported number of input and output axons on the Loihi hardware. Therefore, we execute the SNNs with AVAM method of spiking-MaxPooling on GPUs only. Similar to the case of MJOP Net based SNNs, the individual channels in the preceding Conv layer are flattened and the pooling window's grouped values (in sizes of 4) are passed to the layer of AVAM Nets to estimate the max values; which are then collected, reshaped as channels, and forwarded next. This process is repeated for any additional MaxPooling layers in the network. In the experiments with each of the three architectures, $\phi$ is set to 250Hz, and the *radius* value is varied; Table 6.1 shows the accuracy results.

## 6.4 Discussion

### 6.4.1 Table 6.1 Result Analysis

To evaluate the efficacy of SNNs with our proposed methods of spiking-MaxPooling, we compare it against the results obtained with "True Max $U$" based SNNs (column TMS) and its `ReLU` based non-spiking counterpart (column NSR, with TensorFlow), as well as with SNNs where MaxPooling layers are replaced with AveragePooling layers after training (column MAS, only *Ensemble*s and associated connections removed in AVAM Net). In case of MNIST, the MJOP based SNNs perform similar to their non-spiking counterpart and "True Max $U$" based SNN across all the architectures. In case of FMNIST, the performance drop of MJOP based SNNs is noticeable; and in case of CIFAR10, they perform very poorly. One reason for the accuracy drop is the Loihi hardware constraints i.e. 8-bit quantization of the network weights and fixed-point arithmetic. We found the other reason in case of CIFAR10 to be the highly varying ISI distribution (of Conv layer neurons) across the test images – possibly due to color channels (resulting in highly variable neuron activations). This prevented the choice of a *scale* value to generalize well across all the test images. For MNIST (and to a large extent for FMNIST), we found the ISI distribution of Conv layer neurons to be light-tailed and mostly similar across the test images. More details on the ISI distribution in supplementary material (Appendix A). In a separate experiment, setting two different *scale* values (2 and 1.5) for the corresponding MaxPooling layers in Architecture 2 for FMNIST did not improve the results. Overall, although the MJOP based SNNs were successfully deployed on Loihi, the MJOP spiking-MaxPooling seems suboptimal due to its poor generalizability. For its optimal performance, the maximally firing neuron in each pooling window (in a preceding Conv layer) should have (nearly) same ISI for a consistent effect of *scale* value. On the other hand, AVAM based SNNs perform at par with their non-spiking counterpart and the "True Max $U$" based SNN across all the three datasets and architectures. An important distinction between the MJOP & AVAM methods is that the MJOP method estimates the "True Max $U$" indirectly by scaling the $U_{out}$ (note that this $U_{out}$ *corresponds* to the maximum input $U_i$), whereas the AVAM method estimates the "True Max $U$" directly from the group of $U_i$, which makes the AVAM spiking-MaxPooling more robust and effective; and its optimal performance with the same set of *radius* values across all the three datasets and architectures promises its generalizability. It is interesting to note that in some cases AVAM based SNNs perform better than their non-spiking counterpart and/or "True Max $U$" based SNNs. Also, the SNNs with AveragePooling layers (column MAS) perform poorly compared to all other networks.

### 6.4.2 Adapting MJOP & AVAM spiking-MaxPooling

Our proposed methods of spiking-MaxPooling are suitable for the rate-based SNNs which represent activations as currents (i.e. filtered/synapsed spikes). SNNs which do not filter the spike trains and work directly on binary spikes [98, 34, ...], cannot adapt our proposed methods; this holds true for Time-To-First-Spike based SNNs as well. With respect to the scalability of the MJOP and AVAM Net against the size of pooled inputs, it is linear. For an $r \times c$ MaxPooling window (where $r, c \in \mathbb{N}$), the number of compartments/neurons required in the MJOP and AVAM Net is $r \times c$ and $2 \times (r \times c) - 2$ respectively. However, with the increase in number of neurons in the hierarchical AVAM Net, the estimated max output may get noisier; although, this doesn't hold true for MJOP based method. AVAM Net based spiking-MaxPooling is deployable on any neuromorphic hardware that supports weighted connections and spiking neurons. In our MJOP based SNNs experiments, each channel of a Conv layer (prior to a MaxPooling layer) was small enough in dimensions such that the flattened vector was of size $\leq 1024$, thus easily mapped to a Neuro-Core. If the channel's dimensions are sufficiently large and the flattened vector's size is $> 1024$, then one needs to spatially split the channel and properly map it to more than one Neuro-Core such that no pooling window (thus the corresponding MC neuron) spans across two or more Neuro-Cores, as Loihi restricts the creation of a MC neuron on one Neuro-Core only. However, such a procedure need not be followed with AVAM based SNNs (if deployed on GPUs).

## 6.5 Conclusion

To our best knowledge, this work is a first to present two different hardware-friendly methods of spiking-MaxPooling operation in SNNs, with their evaluation on Loihi. In a first, we also deployed SNNs with MaxPooling layers (via MJOP method) on the Loihi boards. For the appropriate choice of tunable *scale* and *radius* values in MJOP & AVAM Net respectively, we also presented a heuristic method. The Proof-of-Concept Demonstrations on Loihi show that our work makes successful strides towards providing a solution to the MaxPooling problem in SNNs. This also opens up avenues for building hardware ready SNNs with MaxPooling layers without compromising on the quality (otherwise due to replacing MaxPooling with AveragePooling). One immediate future direction of our work is to evaluate the deployment of AVAM based SNNs on a neuromorphic hardware. Next, we hope that our work encourages efforts towards developing other hardware-friendly methods of spiking-MaxPooling.

# Chapter 7

# 3D-CNNs based SNN for Driving Scene Understanding

In this chapter, we leverage the findings in Chapter 4 and 6 to build a 3D-CNN based SNN with spiking-MaxPooling layers for Driving Scene Understanding. I use AVAM Net instead of MJOP Net to implement spiking-MaxPooling in the SNN built because AVAM Net is better performing, hardware independent, more generalizable and robust than MJOP Net. Although the aim was to deploy such an SNN on Loihi boards, I was unable to do so because of the present limitation of Loihi hardware (section 6.3.4) - in the context of AVAM Net based SNNs. As well as, with the MJOP Net, it would have been very challenging to tune the *scale* parameter, thus producing suboptimal results; apart from the technical/design complications in deploying such a large SNN ($\approx$ 10M parameters or more) on the limited cloud-access to the Loihi boards. It was therefore decided to deploy the SNN on GPU.

AVAM based SNNs take significant amount to time to compile, possibly due to extra large number of connections in the AVAM Net spiking-MaxPooling layers. In Chapter 6, the AVAM based SNNs for image classification with Architecture 1 (Fig. 6.7a), 2 (Fig. 6.7b), and 3 (Fig. 6.7c) took around 8 minutes, 35 minutes, and 7 hours respectively to compile. Note that the compile time depends on the depth of the network (and the number of MaxPooling layers) and the size of the input activation maps, and not just on the number of model parameters. Therefore, I trained the 3D-CNNs based ANN (Fig. 4.1) on the HDD dataset with the smallest video-input resolution of $36 \times 64$ frame size and 16 frames temporal depth; total number of trainable parameters $\approx$ 10M. Upon converting it to an SNN and replacing the MaxPooling layers with AVAM Net spiking-MaxPooling, it unfortunately did not compile in reasonable time (it ran for weeks - close to a month). Thus,

although the AVAM based SNN is theoretically entirely deployable on a neuromorphic/non-neuromorphic hardware, I was unable to produce the AVAM based SNN's results for the DSU task.

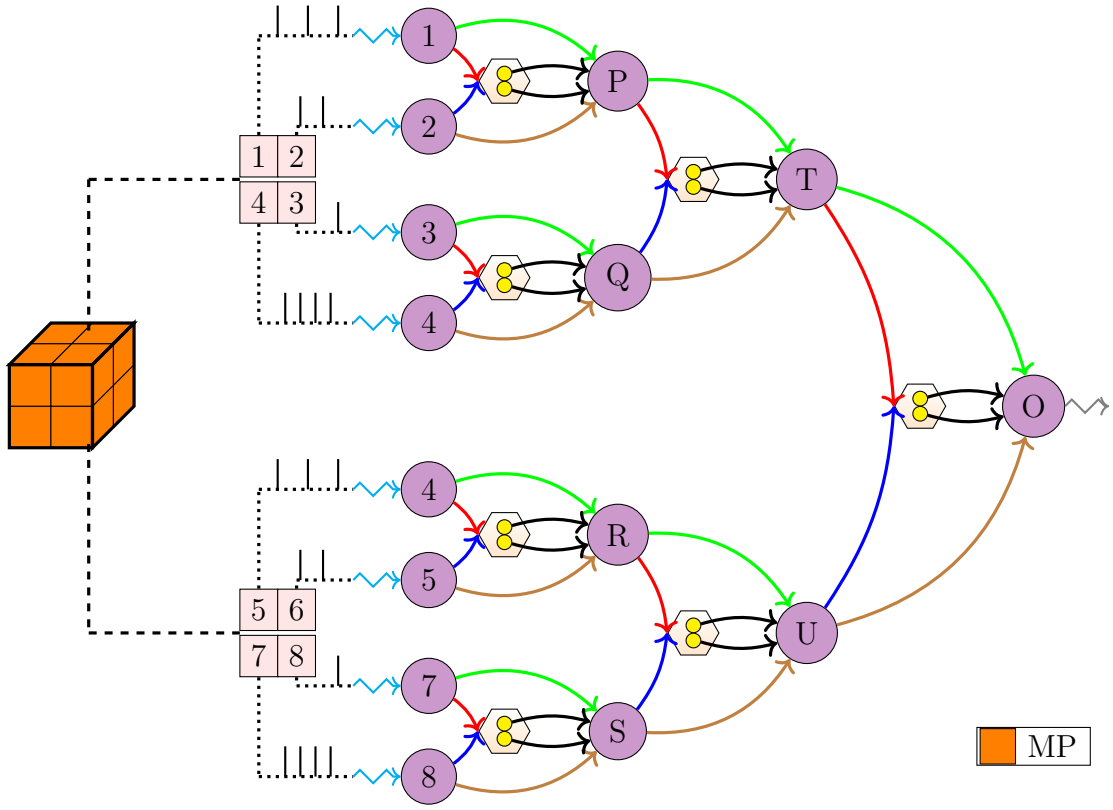



Figure 7.1: *AVAM Net for* $2 \times 2 \times 2$ *MaxPooling.* MP: $2 \times 2 \times 2$ MaxPooling Window. Edges are color coded; refer Fig. 6.3b legend.

## 7.1 Proof of Concept Demonstration

However, I present a Proof-Of-Concept Demonstration of the AVAM Net for $2 \times 2 \times 2$ spiking-MaxPooling (Fig. 7.1, in the 3D-CNNs based SNN) executed on a Loihi chip. The ISI of the 8 spiking inputs are $10, 8, 4, 6, 10, 8, 4, 6$. Three instances of this AVAM Net are executed with IF neurons $\phi$ fixed to 250Hz and $r \in \{0.10, 0.15, 0.20\}$. The estimated $max(U1, U2, U3, U4, U5, U6, U7, U8)$ i.e. "AVAM Net $U_{out}$" (for a corresponding $r$) is shown in the Fig. 7.2. We see that for a value of $r = 0.10$, the hierarchical AVAM Net (in

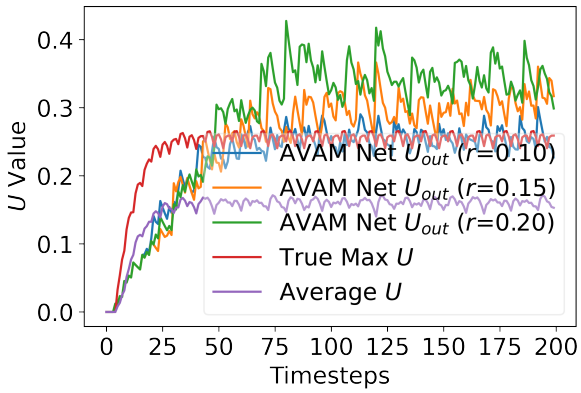



Figure 7.2: "AVAM Net $U_{out}$" are the outputs from the Node `0` in Fig. 7.1 for varying radii $r$. For $r = 0.10$, the $U_{out}$ (blue) very closely matches the "True Max $U$" (red). $\phi = 250$Hz.

Fig. 7.1) very closely approximates the "True Max U"; as well as, the estimated max $U$ are higher than the Average $U$.

## 7.2 Results with SNNs for DSU

Table 7.1 and 7.2 shows the results for experiments with the HDD dataset Layer 0 and 1 respectively - $36 \times 64$ sized frames and 16 frames temporal depth. Note that the settings in section 4.2.3 were used to train the 3D-CNNs based ANNs. Row NSR-MP (and NSR-AP) results are obtained with `ReLU` based non-spiking counterpart ANN with MaxPooling (and AveragePooling). Results with SNNs where non-spiking MaxPooling is done (i.e. "True Max $U$" based SNNs) and where AveragePooling is done (i.e. AveragePooling based SNNs) are shown in rows TMS-x and APS-x respectively (where `x` is the test data presentation time-steps i.e. 40 and 60). In the Table 7.1 and 7.2, it can be seen that experiments with NSR-MP achieve better scores than NSR-AP, thus, MaxPooling is better than AveragePooling in this context. Experiments with TMS-60 achieve a closer result to NSR-MP (and overall a better mAP score in Table 7.1); it is evident that increasing the presentation time-steps helps achieve better scores. Note that in Table 7.1, experiments with APS-x fail to produce reasonable scores on the test data; however, increasing the presentation time-steps might help. Given the successful Proof of Concept Demonstration (Fig. 7.2) for $2 \times 2 \times 2$ MaxPooling with the hierarchical AVAM Net in Fig. 7.1, entirely spiking AVAM based SNNs (with a suitable choice of *radius* and max firing rate $\phi$) can achieve the NSR-MP based scores with a sufficient enough test data presentation time-steps. Al-

though, as mentioned earlier (in the second paragraph of this chapter), this AVAM based SNN for DSU couldn't be executed on a GPU in reasonable time - owing to the modified 3D-CNNs based SNN's high computational complexity due to the large number of connections in the AVAM spiking-MaxPooling layers. This is one evident limitation of AVAM spiking-MaxPooling which needs to be addressed in a future work.

| | **AP results: Layer 0 - Goal Oriented Action Layer** | | | | | | | | | | | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Methods** | **Right Turn** | **Itr. Pass.** | **Merge** | **Left Lane Change** | **Right Lane Branch** | **Right Lane Change** | **Left Turn** | **Crs Pass.** | **Rail. Pass.** | **Left Lane Branch** | **U-Turn** | |
| **NSR-MP** | 67.85 | 71.72 | 4.43 | 35.13 | 1.33 | 28.45 | 61.25 | 8.33 | 0.28 | 25.26 | 14.34 | 28.94 |
| **TMS-40** | 57.28 | 51.73 | 5.87 | 26.30 | 0.50 | 25.18 | 51.02 | 7.31 | 0.21 | 24.57 | 16.53 | 24.23 |
| **TMS-60** | 63.97 | 65.35 | 7.11 | 35.26 | 1.00 | 35.21 | 59.76 | 9.39 | 0.21 | 33.23 | 20.21 | 30.06 |
| **NSR-AP** | 63.10 | 68.62 | 4.73 | 27.77 | 0.80 | 19.39 | 60.14 | 6.79 | 0.30 | 9.86 | 13.92 | 25.04 |
| **APS-40** | 2.30 | 6.58 | 0.12 | 0.43 | 0.12 | 0.45 | 2.62 | 0.15 | 0.04 | 0.17 | 0.50 | 1.23 |
| **APS-60** | 18.02 | 37.90 | 1.77 | 6.50 | 0.15 | 6.65 | 29.74 | 1.07 | 0.07 | 3.96 | 0.57 | 9.67 |

Table 7.1: Average Precision (AP) results for HDD Layer 0 obtained from different SNNs. **Itr. Pass.**: Intersection Passing; **Crs. Pass.**: Crosswalk Passing; **Rail. Pass.**: Railroad Passing; **mAP**: mean Average Precision. **NSR-MP**: **N**on-**S**piking **R**eLU based ANN with **M**ax**P**ooling layers, **NSR-AP**: **N**on-**S**piking **R**eLU based ANN with **A**verage **P**ooling layers, **TMS-x**: "**T**rue **M**ax $U$" based **S**NNs with MaxPooling layers, **APS-x**: **A**verage**P**ooling based **S**NNs (obtained from ANN with AveragePooling layers). **x** in **TMS-x** and **APS-x** denote the presentation time-steps of the test data; here $\mathbf{x} = 40, 60$.

| | **AP results: Layer 1 - Cause Layer** | | | | | | **mAP** |
|---|---|---|---|---|---|---|---|
| **Methods** | **Congestion** | **Sign** | **Traffic Light** | **Crossing Vehicle** | **Parked Car** | **Pedestrian** | |
| **NSR-MP** | 62.51 | 27.19 | 63.35 | 15.71 | 2.91 | 2.86 | 29.09 |
| **NSR-AP** | 64.07 | 23.82 | 52.51 | 11.49 | 4.21 | 3.73 | 26.64 |
| **TMS-40** | 33.54 | 23.87 | 44.04 | 6.17 | 2.59 | 1.58 | 18.63 |
| **TMS-60** | 51.60 | 30.41 | 54.37 | 8.14 | 4.35 | 2.28 | 25.19 |

Table 7.2: Average Precision (AP) results for HDD Layer 1 obtained from different SNNs. **mAP**: mean Average Precision. **NSR-MP**: **N**on-**S**piking **R**eLU based ANN with **M**ax**P**ooling layers, **NSR-AP**: **N**on-**S**piking **R**eLU based ANN with **A**verage **P**ooling layers, **TMS-x**: "**T**rue **M**ax $U$" based **S**NNs with MaxPooling layers. **x** in **TMS-x** denote the presentation time-steps of the test data; here $\mathbf{x} = 40, 60$.

# Chapter 8

# Conclusion and Future Work

This thesis presented a compilation of work necessary to develop an energy efficient SNN for Driving Scene Understanding task. After setting the motivation in Chapter 1 and the necessary background in Chapter 2, Chapter 4 introduced a 3D-CNNs based ANN for DSU (contribution 1 in section 1.2), along with some insights in the necessary spatial and temporal resolution for a new state-of-the-art results on the HDD dataset (contribution 2 in section 1.2). Contribution 1 reinforced the importance of 3D-CNNs over Conv+LSTM based architectures for DSU. Contribution 2 empirically provided a set of initial values of the input video's spatial resolution and temporal depth parameters for DSU; this would be helpful to other researchers to begin their experiments with. Note that it is not necessary that our proposed values of the input video's parameters would be successfully applied right out of the box to other datasets and models, however, our extensive experiments provide a strong foundation for the choice of these parameters which can be tuned later. Contribution 3 provided a new metric "ASiST@x" which overcomes the limitation of the widely used metrics e.g. Average Precision, Accuracy, etc. to measure not only when a particular scene was recognized, but also if the scene recognition was continuous.

However, in accordance with the ANN-to-SNN conversion method, an SNN - entirely deployable on a neuromorphic hardware couldn't be developed due to the absence of spiking versions of the MaxPooling layers in the 3D-CNNs based ANN. Therefore, Chapter 6 introduced two novel methods of spiking-MaxPooling in a first which are entirely deployable on Loihi (contribution 4 in section 1.2). These two methods i.e. MJOP and AVAM are linearly scalable with the size of MaxPooling window, and applicable to any Conv net (e.g. 2D-CNNs, 3D-CNNs). However, tuning the MJOP nets becomes a challenge when the image dataset has color channels and the AVAM net's output gets noisier with the increase in MaxPooling size. Nonetheless, the SNNs incorporating these spiking-MaxPooling meth-

ods achieved comparable results to their ANN counterpart on the image classification task (contribution 5 in section 1.2). In a first, these SNNs with spiking-MaxPooling layers were also deployed on Loihi - a neuromorphic hardware; so far none of the available MaxPooling methods in SNNs were successful to do so. Bringing together the foundations from Chapter 4 and Chapter 6, Chapter 7 attempted to build an SNN with AVAM spiking-MaxPooling - entirely deployable on a neuromorphic hardware (contribution 6 in section 1.2); however, due to very high compile time of AVAM based SNNs, it couldn't be evaluated.

The immediate future work would be to explore approaches for lowering down the large number of extra connections in the AVAM spiking-MaxPooling layers; thereby lowering down the compilation time. Apart from this, one can also look for other efficient methods for spiking-MaxPooling. Another important research direction would be to look into lowering down the presentation time-steps of incoming test data for a faster inference time. In section 4.1.2, we reused the C3D architecture's hyper-parameters [111] for our experiments. The kernel size in C3D network - $d \times 3 \times 3$ was varied for $d \in \{1, 3, 5, 7\}$ where $d$ is the temporal depth. Tran et al. [111] found that a value of $d = 3$ performs the best (we use the same) and $d = 1$ performs worst due to the lack of motion modelling. However, certain driving scenarios e.g. U-turn can take significantly longer time to complete. Will a higher value of $d$ be helpful (do note that one needs to consider the frame sampling rate as well while experimenting with $d$)? Similarly, while implementing 3D MaxPooling Tran et al. [111] fixed the pooling temporal depth $p$ to 1 for the first pooling layer, and 2 for the rest of the pooling layers; and also found that not merging the temporal information too early (i.e. $p = 1$ in the first pooling layer) helps to achieve better performance. Will $p = 1$ for the next pooling layers too - in the context of Driving Scene Understanding help? Such an exploration of the kernel's temporal depth $d$ and pooling temporal depth $p$ is left for the future work.

# References


[1] Mohammad Sadegh Aliakbarian, Fatemeh Sadat Saleh, Mathieu Salzmann, Basura Fernando, Lars Petersson, and Lars Andersson. Viena$^2$: A driving anticipation dataset. In *Asian Conference on Computer Vision*, pages 449–466. Springer, 2018.

[2] Rajagopal Ananthanarayanan, Steven K Esser, Horst D Simon, and Dharmendra S Modha. The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses. In *Proceedings of the conference on high performance computing networking, storage and analysis*, pages 1–12, 2009.

[3] Edmund L. Andrews. AI's Carbon Footprint Problem. https://hai.stanford.edu/blog/ais-carbon-footprint-problem.

[4] Go Ashida and Waldo Nogueira. Spike-conducting integrate-and-fire model. *eNeuro*, 5(4), 2018.

[5] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[6] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

[7] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[8] Holger Berndt, Jorg Emmert, and Klaus Dietmayer. Continuous driver intention recognition with hidden markov models. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pages 1189–1194. IEEE, 2008.

[9] Murphy Berzish, Chris Eliasmith, and Bryan Tripp. Real-time fpga simulation of surrogate models of large spiking networks. In *International Conference on Artificial Neural Networks*, pages 349–356. Springer, 2016.

[10] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472, 1998.

[11] Huikun Bi, Ruisi Zhang, Tianlu Mao, Zhigang Deng, and Zhaoqi Wang. How can i see my future? fvtraj: Using first-person view for pedestrian trajectory prediction. In *European Conference on Computer Vision*, pages 576–593. Springer, 2020.

[12] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pages 1–8, 2019.

[13] Marc Bocquet, Hassen Aziza, Weisheng Zhao, Yue Zhang, Santhosh Onkaraiah, Christophe Muller, Marina Reyboz, Damien Deleruyelle, Fabien Clermidy, and Jean-Michel Portal. Compact modeling solutions for oxide-based resistive switching memories (oxram). *Journal of Low Power Electronics and Applications*, 4(1):1–14, 2014.

[14] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

[15] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[16] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *Conference on Robot Learning*, pages 947–956, 2018.

[17] Yang Yin Chen, Robin Degraeve, Sergiu Clima, Bogdan Govoreanu, Ludovic Goux, Andrea Fantini, Gouri Sankar Kar, Geoffrey Pourtois, Guido Groeseneken, Dirk J Wouters, et al. Understanding of the endurance failure in scaled hfo 2-based 1t1r rram through vacancy mobility degradation. In *2012 International Electron Devices Meeting*, pages 20–3. IEEE, 2012.

[18] Xiang Cheng, Yunzhe Hao, Jiaming Xu, and Bo Xu. Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition. In *IJCAI*, pages 1519–1525, 2020.

[19] Elisabetta Chicca, Fabio Stefanini, Chiara Bartolozzi, and Giacomo Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102(9):1367–1388, 2014.

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[21] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[22] Ali Diba, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. Temporal 3d convnets: New architecture and transfer learning for video classification. *arXiv preprint arXiv:1711.08200*, 2017.

[23] Ali Diba, Ali Mohammad Pazandeh, and Luc Van Gool. Efficient two-stream motion and appearance 3d cnns for video classification. *arXiv preprint arXiv:1608.08851*, 2016.

[24] Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[25] Jieyun Ding, Ruina Dang, Jianqiang Wang, and Keqiang Li. Driver intention recognition method based on comprehensive lane-change environment assessment. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 214–220. IEEE, 2014.

[26] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[27] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems*, 28:1117–1125, 2015.

[28] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the national academy of sciences*, 113(41):11441–11446, 2016.

[29] Davi Frossard, Eric Kee, and Raquel Urtasun. Deepsignals: Predicting intent of drivers through visual signals. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9697–9703. IEEE, 2019.

[30] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[31] Isha Garg, Sayeed Shafayet Chowdhury, and Kaushik Roy. Dct-snn: Using dct to distribute spatial information over time for learning low-latency spiking neural networks. *arXiv preprint arXiv:2010.01795*, 2020.

[32] Ramashish Gaurav, Bryan Tripp, and Apurva Narayan. Driving scene understanding: How much temporal context and spatial resolution is necessary? *Proceedings of the Canadian Conference on Artificial Intelligence*, 6 2021. https://caiac.pubpub.org/pub/d195apoz.

[33] B Govoreanu, GS Kar, YY Chen, V Paraschiv, S Kubicek, A Fantini, IP Radu, L Goux, S Clima, R Degraeve, et al. 10× 10nm 2 hf/hfo x crossbar resistive ram with excellent performance, reliability and low-energy operation. In *2011 International Electron Devices Meeting*, pages 31–6. IEEE, 2011.

[34] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 388–404. Springer, 2020.

[35] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*, 2019.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] Michael Hennecke, Wolfgang Frings, Willi Homberg, Anke Zitz, Michael Knobloch, and Hans Böttiger. Measuring power consumption on ibm blue gene/p. *Computer Science-Research and Development*, 27(4):329–336, 2012.

[38] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[39] Haijing Hou, Lisheng Jin, Qingning Niu, Yuqin Sun, and Meng Lu. Driver intention recognition method using continuous hidden markov model. *International Journal of Computational Intelligence Systems*, 4(3):386–393, 2011.

[40] Yangfan Hu, Huajin Tang, and Gang Pan. Spiking deep residual network. *arXiv preprint arXiv:1805.01352*, 2018.

[41] Yuhuang Hu and Michael Pfeiffer. Max-pooling operations in deep spiking neural networks, 2016.

[42] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, 17(4):879–892, 2006.

[43] Xiaoyu Huang, Edward Jones, Siru Zhang, Shouyu Xie, Steve Furber, Yannis Goulermas, Edward Marsden, Ian Baistow, Srinjoy Mitra, and Alister Hamilton. An fpga implementation of convolutional spiking neural networks for radioisotope identification. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[44] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[45] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[46] Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*, 2016.

[47] Nabil Imam and Thomas A Cleland. Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nature Machine Intelligence*, 2(3):181–191, 2020.

[48] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[49] Weiwen Jiang, Bike Xie, Chun-Chen Liu, and Yiyu Shi. Integrating memristors and cmos for better ai. *Nature Electronics*, 2(9):376–377, 2019.

[50] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, and Sarah Mack. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[51] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.

[52] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

[53] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11270–11277, 2020.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Michael N Kozicki, Maria Mitkova, and Ilia Valov. Electrochemical metallization memories. *Resistive Switching: From Fundamentals of Nanoionic Redox Processes to Memristive Device Applications*, pages 483–514, 2016.

[56] Andrzej S Kucik and Gabriele Meoni. Investigating spiking neural networks for energy-efficient on-board ai applications. a case study in land cover and land use classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2020–2030, 2021.

[57] Souvik Kundu, Gourav Datta, Massoud Pedram, and Peter A Beerel. Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3953–3962, 2021.

[58] Louis Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathololgy*, 9:620–635, 1907.

[59] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[60] Chen Li, Runze Chen, Christoforos Moutafis, and Steve Furber. Robustness to noisy synaptic weights in spiking neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[61] Chengxi Li, Stanley H Chan, and Yi-Ting Chen. Who make drivers stop? towards driver-centric risk assessment: Risk object identification via causal inference. *arXiv preprint arXiv:2003.02425*, 2020.

[62] Chengxi Li, Yue Meng, Stanley H Chan, and Yi-Ting Chen. Learning 3d-aware egocentric spatial-temporal interaction via graph convolutional networks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8418–8424. IEEE, 2020.

[63] Jingling Li, Weitai Hu, Ye Yuan, Hong Huo, and Tao Fang. Bio-inspired deep spiking neural network for image classification. In *International Conference on Neural Information Processing*, pages 294–304. Springer, 2017.

[64] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[65] Zhitao Lin, Juncheng Shen, De Ma, and Jianyi Meng. Quantisation and pooling method for low-inference-latency spiking neural networks. *Electronics Letters*, 53(20):1347–1348, 2017.

[66] Te-Yuan Liu, Ata Mahjoubfar, Daniel Prusinski, and Luis Stevens. Neuromorphic computing for content-based image retrieval. *arXiv preprint arXiv:2008.01380*, 2020.

[67] Paolo Livi and Giacomo Indiveri. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. In *2009 IEEE international symposium on circuits and systems*, pages 2898–2901. IEEE, 2009.

[68] Hin Wai Lui and Emre Neftci. Hessian aware quantization of spiking neural networks. *arXiv preprint arXiv:2104.14117*, 2021.

[69] Timothée Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS computational biology*, 3(2):e31, 2007.

[70] Rob Matheson. Reducing the carbon footprint of artificial intelligence. https://news.mit.edu/2020/artificial-intelligence-ai-carbon-footprint-0423.

[71] Mark D McDonnell, Kwabena Boahen, Auke Ijspeert, and Terrence J Sejnowski. Engineering intelligent electronic systems based on computational neuroscience [scanning the issue]. *Proceedings of the IEEE*, 102(5):646–651, 2014.

[72] Sam McKennoch, Dingding Liu, and Linda G Bushnell. Fast modifications of the spikeprop algorithm. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 3970–3977. IEEE, 2006.

[73] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[74] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[75] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier. Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Frontiers in neuroscience*, 13:625, 2019.

[76] Robert A Nawrocki, Richard M Voyles, and Sean E Shaheen. A mini review of neuromorphic architectures and implementations. *IEEE Transactions on Electron Devices*, 63(10):3819–3829, 2016.

[77] Alexander Neckar, Sam Fok, Ben V Benjamin, Terrence C Stewart, Nick N Oza, Aaron R Voelker, Chris Eliasmith, Rajit Manohar, and Kwabena Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, 2018.

[78] Emre O Neftci. Data and power efficient intelligence with neuromorphic learning machines. *Iscience*, 5:52–68, 2018.

[79] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[80] Duy-Anh Nguyen, Xuan-Tu Tran, Khanh N Dang, and Francesca Iacopi. A lightweight max-pooling method and architecture for deep spiking convolutional neural networks. In *2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 209–212. IEEE, 2020.

[81] World Health Organization. Dept. of Violence, Injury Prevention, World Health Organization. Violence, Injury Prevention, and World Health Organization. *Global status report on road safety: time for action*. World Health Organization, 2009.

[82] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. Hfirst: A temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2028–2040, 2015.

[83] World Health Organization et al. Global status report on road safety 2018: summary. Technical report, World Health Organization, 2018.

[84] Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[85] Kinjal Patel, Eric Hunsberger, Sean Batir, and Chris Eliasmith. A spiking neural network for image segmentation. *arXiv preprint arXiv:2106.08921*, 2021.

[86] Prajakta Ganesh Pawar and V Devendran. Scene understanding: A survey to see the world at a single glance. In *2019 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, pages 182–186. IEEE, 2019.

[87] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018.

[88] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[89] Vasili Ramanishka, Yi-Ting Chen, Teruhisa Misu, and Kate Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7699–7707, 2018.

[90] Venkat Rangan, Abhishek Ghosh, Vladimir Aparin, and Gert Cauwenberghs. A subthreshold avlsi implementation of the izhikevich simple neuron model. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 4164–4167. IEEE, 2010.

[91] Daniel Rasmussen. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, 17(4):611–628, 2019.

[92] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[93] Nele Rietmann, Beatrice Hügler, and Theo Lieven. Forecasting the trajectory of electric vehicle sales and the consequences for worldwide co2 emissions. *Journal of Cleaner Production*, 261:121038, 2020.

[94] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[95] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950. IEEE, 2010.

[96] Benjamin Schrauwen, Jan Van Campenhout, et al. Improving spikeprop: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC workshop*, volume 11, pages 301–305, 2004.

[97] Andreas Th Schulz and Rainer Stiefelhagen. Pedestrian intention recognition using latent-dynamic conditional random fields. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 622–627. IEEE, 2015.

[98] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[99] Saima Sharmin, Priyadarshini Panda, Syed Shakib Sarwar, Chankyu Lee, Wachirawit Ponghiran, and Kaushik Roy. A comprehensive analysis on adversarial robustness of spiking neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[100] Saima Sharmin, Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations. In *European Conference on Computer Vision*, pages 399–414. Springer, 2020.

[101] Juncheng Shen, De Ma, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. Darwin: a neuromorphic hardware co-processor based on spiking neural networks. *Science China Information Sciences*, 59(2):1–5, 2016.

[102] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.

[103] Terrence C Stewart. A technical overview of the neural engineering framework. *University of Waterloo*, 110, 2012.

[104] Terrence C Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the nengo simulator. *Frontiers in neuroinformatics*, 3:7, 2009.

[105] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in neuroscience*, 9:222, 2015.

[106] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[107] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[108] Guangzhi Tang, Arpit Shah, and Konstantinos P Michmizos. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4176–4181. IEEE, 2019.

[109] Johannes C Thiele, Olivier Bichler, and Antoine Dupret. Event-based, timescale invariant unsupervised online deep learning with stdp. *Frontiers in computational neuroscience*, 12:46, 2018.

[110] Martin Torstensson, Boris Duran, and Cristofer Englund. Using recurrent neural networks for action and intention recognition of car drivers. In *8th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2019, Prague, Czech Republic; 19-21 February, 2019*, pages 232–242. SciTePress, 2019.

[111] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[112] Henry Clavering Tuckwell. *Introduction to theoretical neurobiology: linear cable theory and dendritic structure*, volume 1. Cambridge University Press, 1988.

[113] Xiao-Jing Wang and György Buzsáki. Gamma oscillation by synaptic inhibition in a hippocampal interneuronal network model. *Journal of neuroscience*, 16(20):6402–6413, 1996.

[114] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80% energy savings. *arXiv preprint arXiv:1910.13349*, 2019.

[115] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[116] Mingze Xu, Mingfei Gao, Yi-Ting Chen, Larry S Davis, and David J Crandall. Temporal recurrent networks for online action detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5532–5541, 2019.

[117] Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10577–10584, 2021.

[118] Bo Zhao, Shoushun Chen, and Huajin Tang. Bio-inspired categorization using event-driven feature extraction and spike-based learning. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 3845–3852, 2014.

# APPENDICES

# Appendix A

# Supplementary material of ISI Plots for Spiking-MaxPooling

## A.1 ISI DISTRIBUTION PLOTS

Herein, we plot the ISI distributions of the Conv layer neurons (random and 2048 in number) of 36 test images of all the three datasets each with respect to the Architectures 1 and 2, to support our claims in the paper. The plots correspond to the Conv layers prior to the MaxPooling layers only and are obtained from "True Max $U$" based SNNs. As mentioned in the paper, for the optimal performance of MJOP Net, the maximally firing neuron in each pooling window should have nearly same ISI for a consistent effect of the *scale* value. If the ISIs of maximally firing neurons are very different, a single *scale* value doesn't generalize well (as the *scale* value is dependent on $U_i$).

Therefore, for the datasets where ISI distributions are light-tailed and mostly similar across the test images, a suitable choice of *scale* value generalizes well (across the test images), thus desirable performance of MJOP based SNNs. Light-tailed distribution implies that most of the neurons fire at lower ISIs and those ISIs are nearly same (as will be seen in the case of MNIST and to a large extent in FMNIST dataset), thus a high probability of (nearly) same ISI of maximally firing neurons in MaxPooling windows. Datasets for which the distributions are fat-tailed and are varied across the test images (as will be seen in case of CIFAR10 dataset), a chosen *scale* value doesn't generalize well; thus making it difficult to tune the *scale* parameter. Moreover, in case of a deeper architecture, a poor approximation of an earlier MaxPooling layer due to a suboptimal *scale* value further degrades the approximation of later MaxPooling layers. Following are the ISI distribution plots:

### A.1.1 Plots for Architecture 1 - MNIST - Conv_1: Fig. A.1


Figure A.1: *Arch.* 1, *MNIST, Conv_1.* ISI distributions are light-tailed and similar.

### A.1.2 Plots for Architecture 1 - FMNIST - Conv_1: Fig. A.2

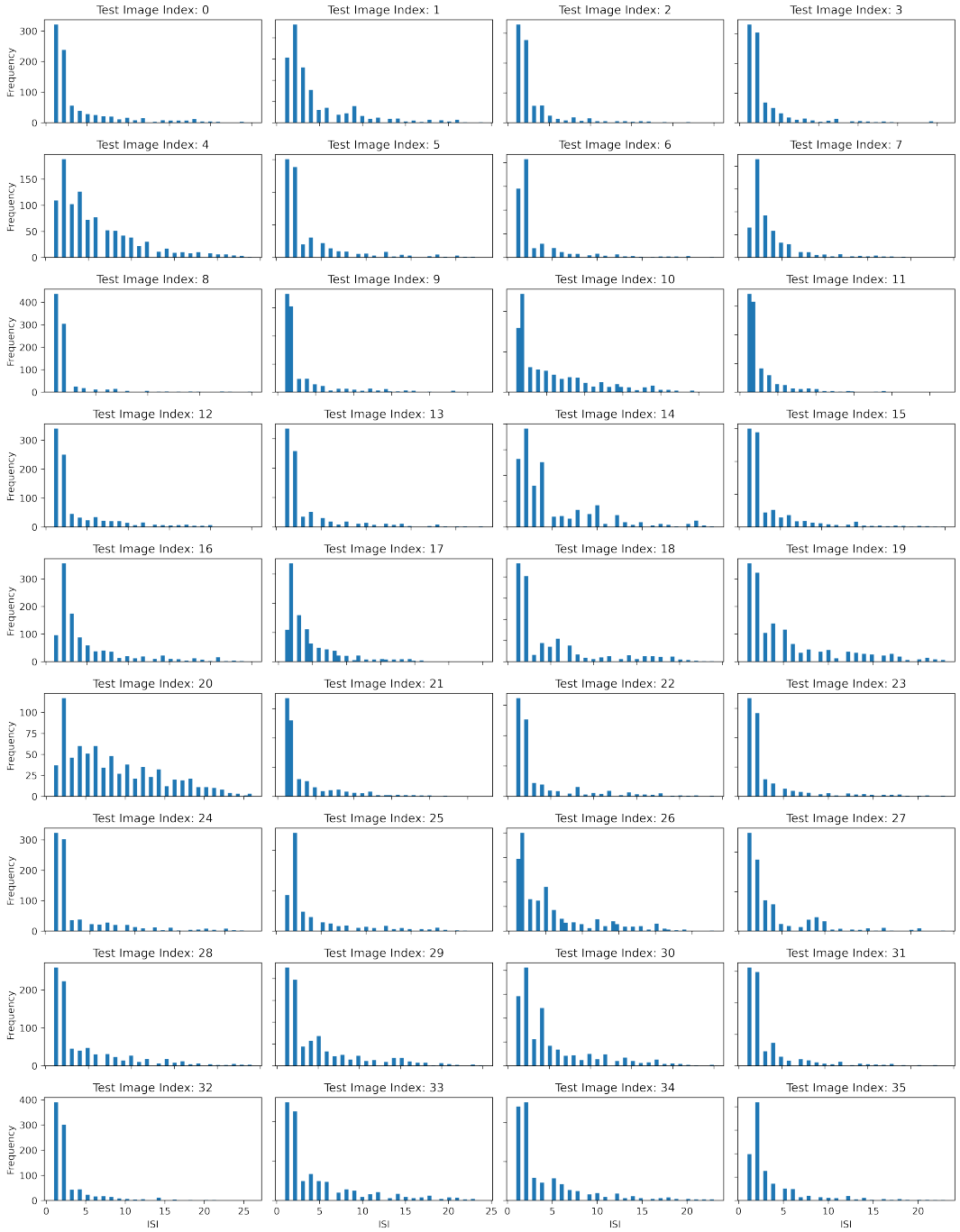



Figure A.2: *Arch.* 1, *FMNIST, Conv_1.* Most ISI distributions are light-tailed and mostly similar.

### A.1.3 Plots for Architecture 1 - CIFAR10 - Conv_1: Fig. A.3

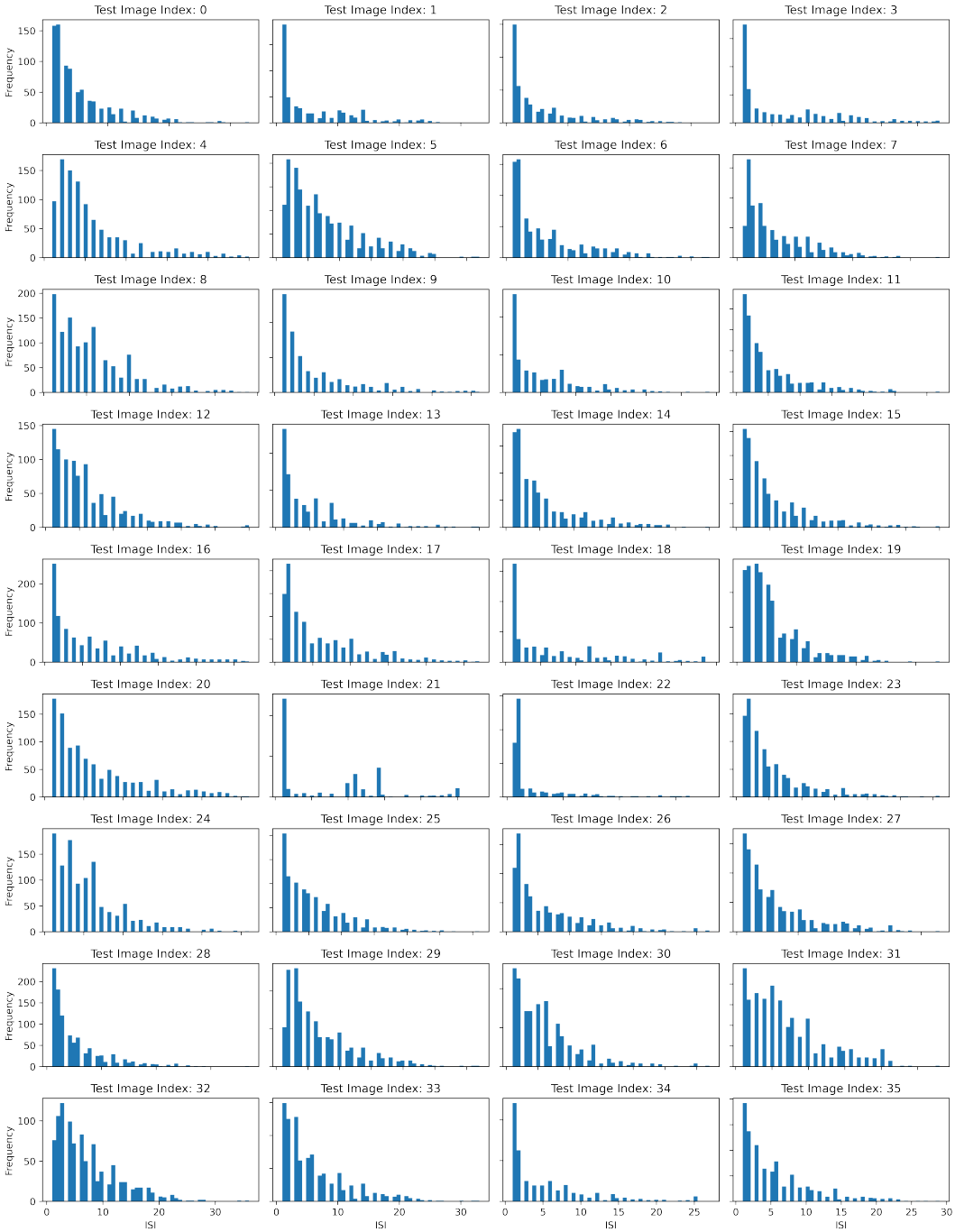



Figure A.3: *Arch.* 1, *CIFAR10, Conv_1.* ISI distributions are dissimilar and mostly fat-tailed.

### A.1.4 Plots for Architecture 2 - MNIST - Conv_1: Fig. A.4


Figure A.4: *Arch.* 2, *MNIST, Conv_1.* ISI distributions are light-tailed and similar.

### A.1.5 Plots for Architecture 2 - MNIST - Conv_2: Fig. A.5

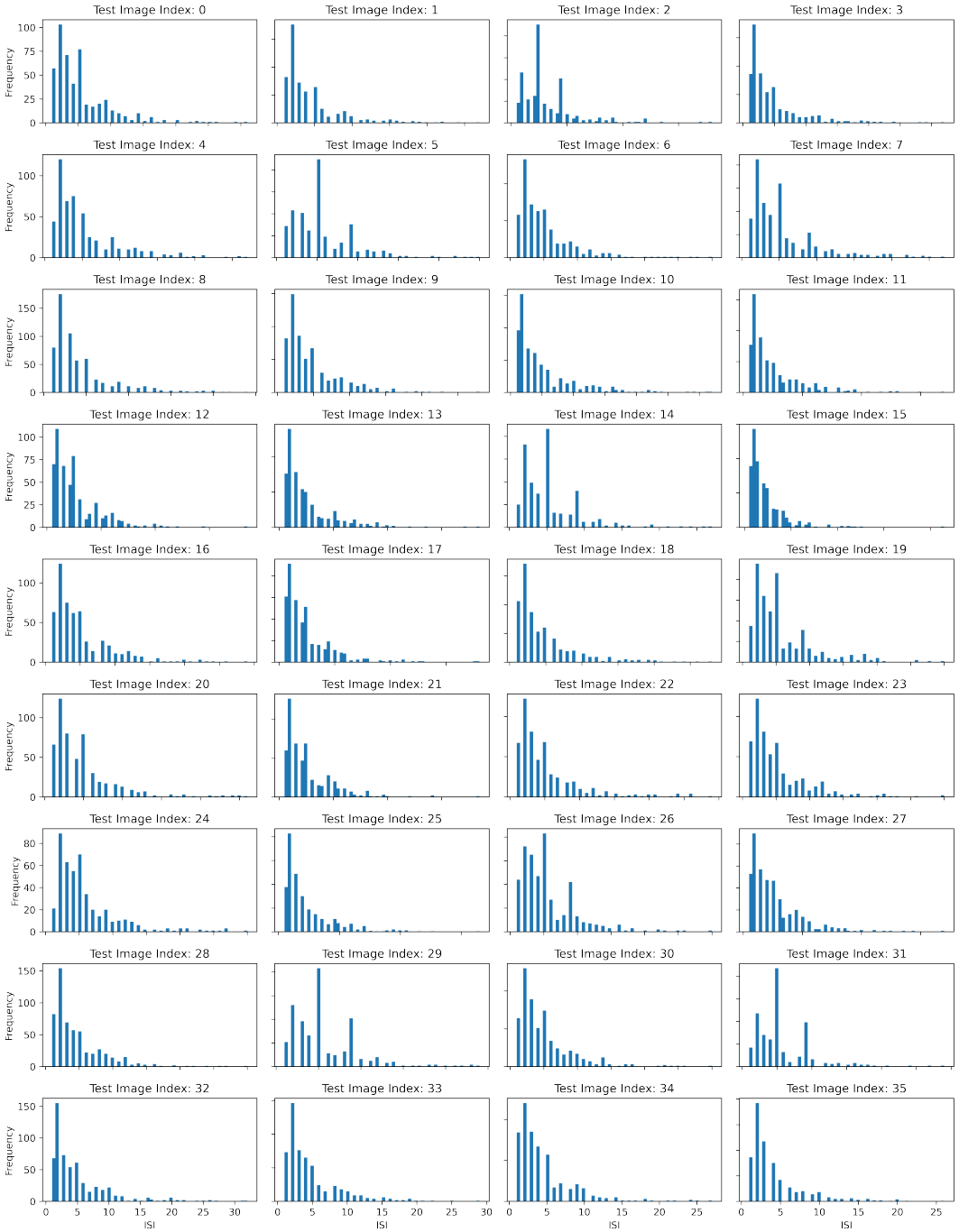



Figure A.5: *Arch.* 2*, MNIST, Conv_2.* ISI distributions are not very similar, and not very light-tailed.

### A.1.6 Plots for Architecture 2 - FMNIST - Conv_1: Fig. A.6

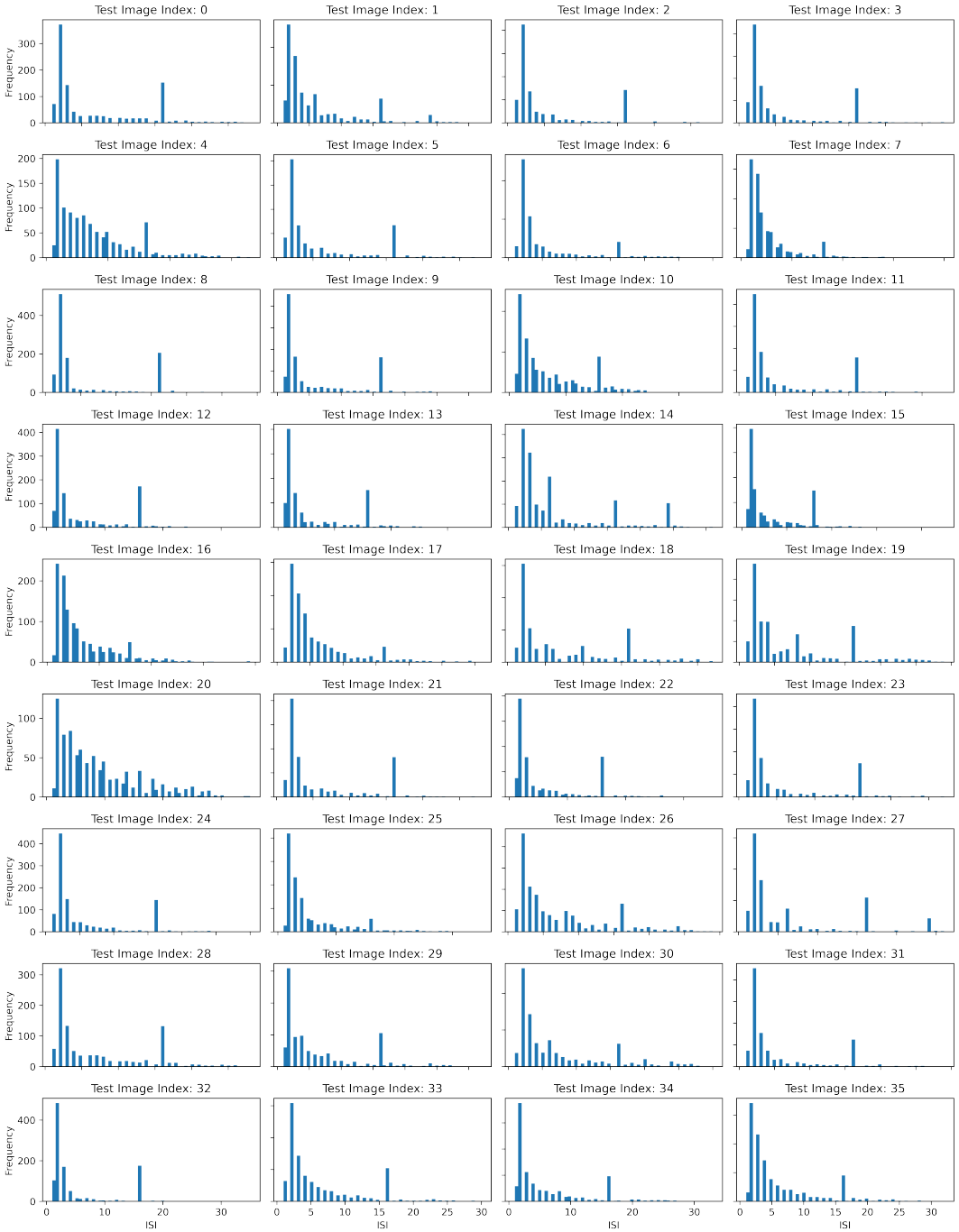



Figure A.6: *Arch. 2, FMNIST, Conv_1.* ISI distributions are almost light-tailed and mostly similar.

### A.1.7 Plots for Architecture 2 - FMNIST - Conv_2: Fig. A.7

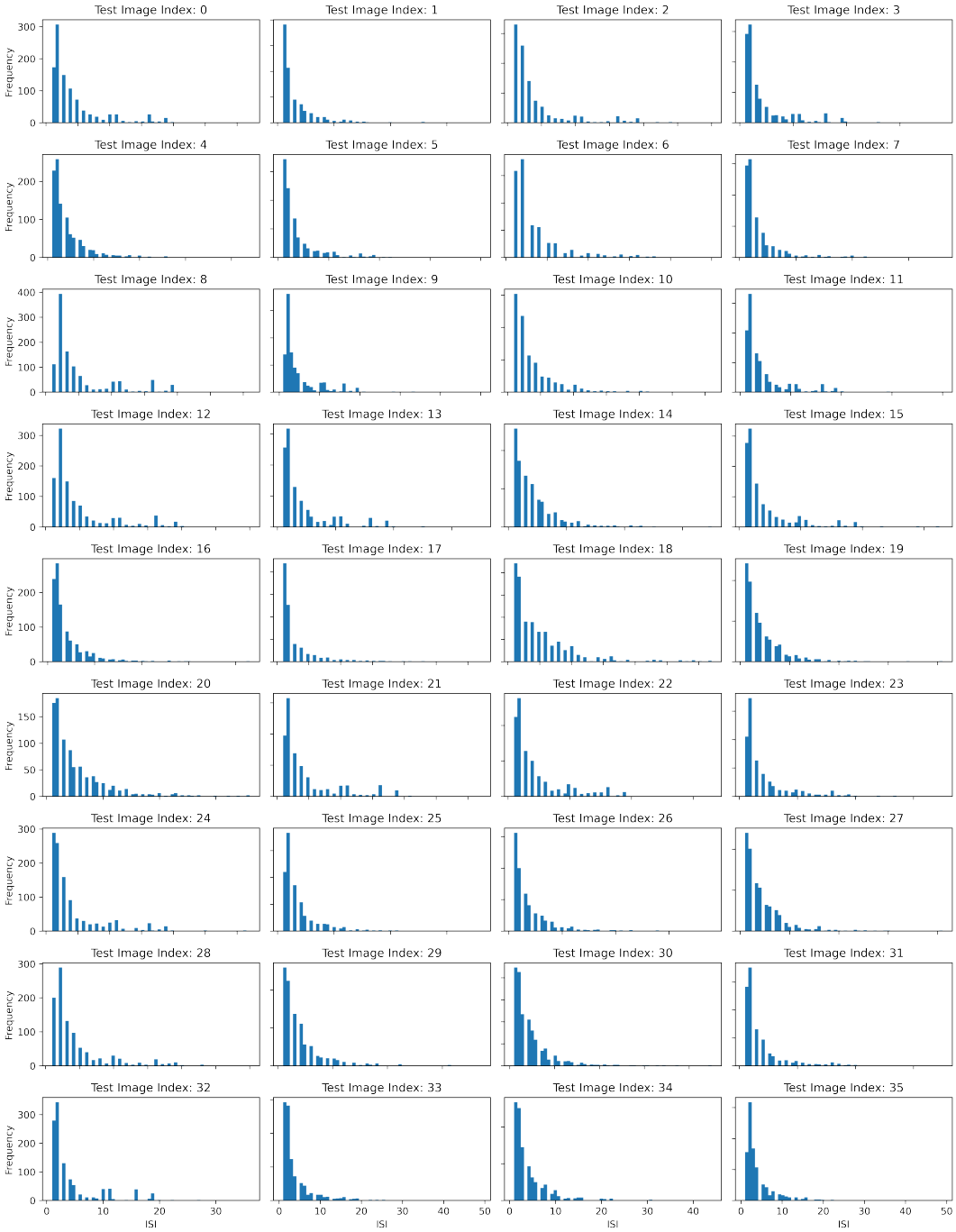



Figure A.7: *Arch.* 2, *FMNIST, Conv_2.* ISI distributions are light-tailed and similar.

### A.1.8 Plots for Architecture 2 - CIFAR10 - Conv_1: Fig. A.8

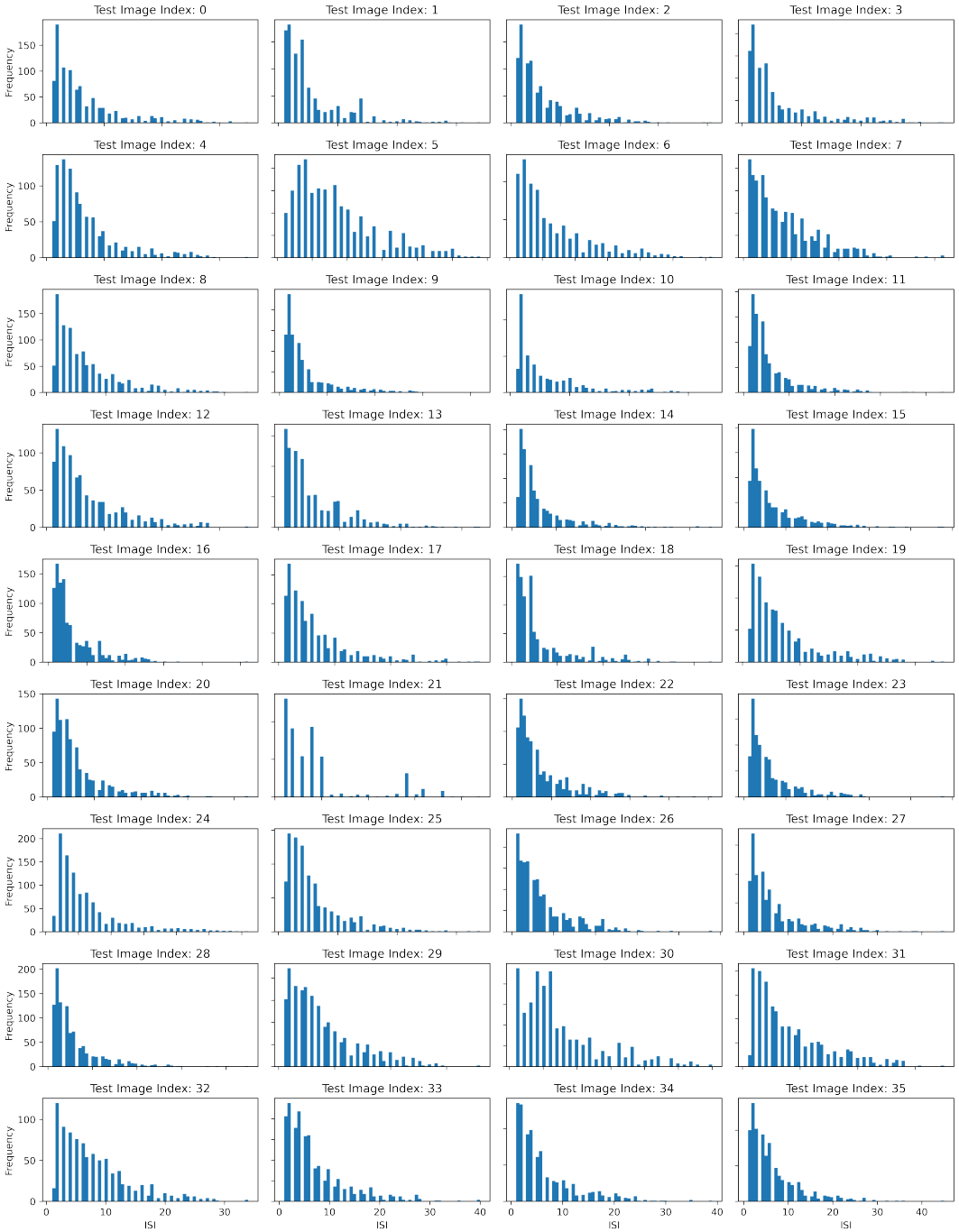



Figure A.8: *Arch. 2, CIFAR10, Conv_1.* ISI distributions dissimilar and fat-tailed.

### A.1.9 Plots for Architecture 2 - CIFAR10 - Conv_2: Fig. A.9

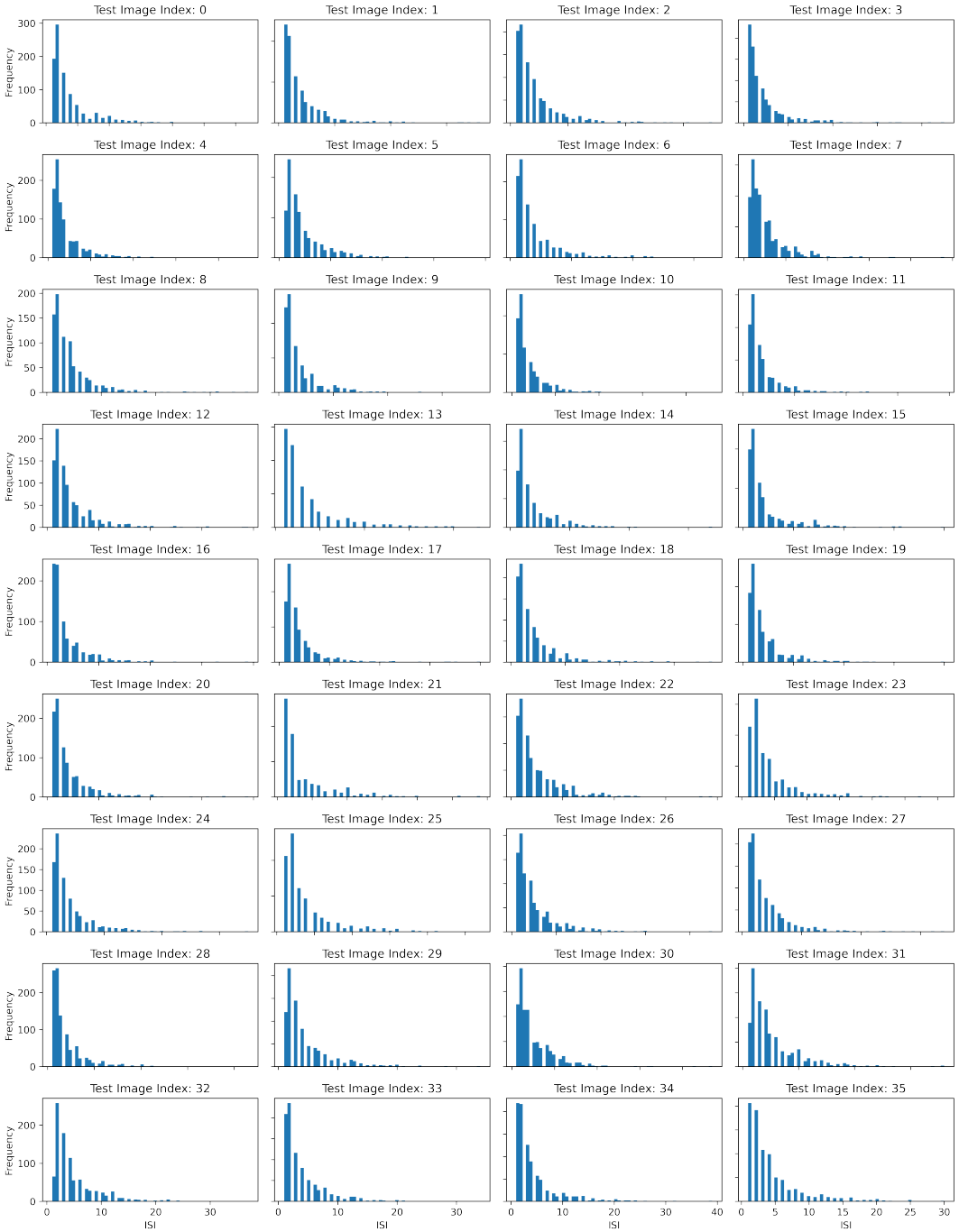



Figure A.9: *Arch.* 2, *CIFAR10, Conv_2.* ISI distributions are light-tailed and similar.