Universiteit Leiden

Master Computer Science

Analysis of the relation between bitdepth and timesteps in ANN to SNN conversion

Name:	Christian Steennis
Student ID:	2636727
Date:	August 29, 2025
Specialisation:	Artificial Intelligence
1st supervisor:	Joost Broekens
2nd supervisor:	Qinyu Chen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Neural network quantization and conversion of artificial neural networks (ANN) to spiking neural networks (SNN) are both methods of lowering the energy consumption of neural networks. Where quantization lowers the precision of a network in the spatial dimension, SNN conversion does the same in the temporal dimension. This similarity in methodology implies there is a relation between the two. By converting ANN to SNN and measuring the accuracy drop relative to the conversion point of the SNN and comparing it to the accuracy drop of the quantized ANN relative to unquantized performance, we get the relation. Resulting in an empirical study showing the known bound $T = 2^b - 1$ is a huge overestimation and determining an empirical relation between bit depth and time steps. The experiments are conducted on varying convolutional models, confining the conclusions to this type of model. It is found that the relation follows a general logistic curve where the function parameters depend on the complexity of the data and models, although more research is needed to determine the right parameter values.

1 Introduction

Artificial Neural Networks (ANN), and Large Language Models (LLMs) in particular, can perform complicated tasks increasingly well. These tasks include audio-visual generation tasks like speech synthesis and image creation. A general drawback of these models is that they perform better when they are larger. This leads to huge models with billions of parameters. Training all of these parameters comes with a large energy consumption. Therefore, energy-efficient training and inference of machine learning models is an active field of research [36, 16].

Spiking Neural Networks (SNN) [27] have been around for a while and are proven to be very energy efficient at inference. This type of neural network has an event-driven nature and brain-like synaptic structures. These special properties enable them to be implemented on neuromorphic hardware chips like Intel's Loihi [6, 30], IBM's TrueNorth [1] and SpiNNaker [14, 28]. Chips like these are developed specifically for hosting SNNs. Additionally, because of their low energy consumption, they can be embedded in systems where energy is a limited resource, e.g., in robotics.

The energy-efficient nature of SNNs makes them a compelling alternative to classic deep learning techniques. Since SNNs are functionally different from classic Artificial Neural Networks (ANN) a conversion is needed. There are two ways to go about doing this, by converting a pre-trained model [11, 33, 4, 31] or by training from scratch [38, 9, 13, 25]. Conversion copies over the ANN's weights and will approximate the original ANN output values. The other method of training an SNN works by constructing a structurally equivalent SNN and training it from scratch on the same data. Both methods are suitable for different situations. There already exist a lot of (large) powerful models for various tasks. For these, converting to energy-efficient counterparts is more useful than training from scratch. For new tasks it might be more economical to design them as an SNN and train them as such. This assumes there are efficient training methods available. That being said, training an SNN is often complicated and time-consuming because computing gradients through time is not trivial. Therefore, since the training costs have already been incurred for existing models, it is often more efficient to transfer the trained weights and activations directly to an SNN. This also eliminates the need for gradient calculation in hardware, keeping the hardware small and efficient.

Not all components of an ANN can be converted easily. Most methods focus on converting the activation values to spike rates by copying the weights [11, 4, 33]. However, this will not work for all components and activation functions. Only the ReLU is well-suited for the conversion, as stated

by [4]. Other common ANN components like LSTM and attention layers are not trivial to convert. Even a commonly converted architecture like a Convolutional Neural Network (CNN) has difficult components like MaxPooling. Therefore, it is common practice to first tailor the original ANN to a version where easily convertible components replace the original components. This method needs retraining in the ANN domain to work effectively.

When lowering the bit precision of an ANN, a drop in accuracy is expected. The same holds when not enough timesteps are used for spiketrains inference in an SNN. Since ANN-to-SNN conversion is analogous to ANN quantization [23, 17], converting a low precision ANN to SNN is a logical step. Quantization is a method for lowering the precision of high-precision data. This technique from signal processing is frequently used in ANNs to decrease memory usage and increase interference times. There are two main methods for ANN quantization, post-training quantization and quantization-aware training. The first method works well for quantization to mid-range bit lengths, however struggles with accuracy for extremely low bit precisions like 1 or 2 bits. These extreme quantization methods need quantization-aware training to achieve usable results. There are frameworks provided for extreme bit quantization like DoReFa-Net [40] and FINN [37]. They have the ability to convert an ANN as far as to a Binarized Neural Network [18]. Converting these extreme quantized neural networks to SNNs leads to a very low number of time steps needed for correct inference.

The number of timesteps an SNN needs to achieve the same accuracy as the ANN does not only depend on the ANN's bit depth but also depends on the task and type of model. There is little to no research on the exact number of timesteps one should use and the relation between quantization and SNN conversion, other than empirical results showing lower inference times [17, 23, 5]. The aim of this research is to investigate the relation between the degradation in performance after (a) quantization and (b) an insufficient number of time steps after conversion. This to give insight into the effect of different models and data on this relation.

2 Background and Related Work

2.1 Spiking Neural Networks

Spiking Neural Networks (SNN) [27] are neural networks that are designed with the biological workings of the brain in mind. They communicate through spikes instead of floating-point decimals. The neurons in an SNN are connected through synapses, which convey electrical spikes. Over time these spikes form a sequence called a spiketrain. In most simulations the spiketrain is an array of ones and zeros, where one is a spike and zero is no spike. The neurons of an SNN implement an Integrate-and-Fire (IF) neuron, optionally leaky (LIF) [21]. These neurons have an internal membrane potential that is increased when a spike arrives. When the potential $V_i^l(t)$ meets a certain threshold V_{th} the neuron emits a spike. When this occurs, the potential is reset to zero or subtracted by the threshold [33]. In the leaky variant, the membrane potential is decreased over time. This allows for SNNs to adopt temporal dependencies in data. $v_i^l(t)$ increases with the sum of the input spikes coming from the previous layer. The membrane potential $v_i^l(t)$ is updated as follows

$$v_i^l(t) = v_i^l(t-1) + z_i^l(t) - V_{th}\theta_i^l(t)$$

where $z_i^l(t)$ is the input current to the i-th neuron in the l-th layer. $z_i^l(t)$ is calculated using the following equation, where a_j^{l-1} is the input from the j-th neuron in the previous layer

$$z_i^l(t) = \sum_{j=1}^{M^{l-1}} W_{ij}^l a_j^{l-1}(t) + b_i^l$$

$\theta_i^l(t)$ controls whether a neuron fires a spike, which only happens when the threshold is met or exceeded. When this happens, the neuron's membrane potential is reset by subtraction of the threshold, as in [33].

$$\theta_i^l(t) = \begin{cases} 1 & \text{if } v_i^l(t-1) + z_i^l(t) \geq V_{th} \\ 0 & \text{else} \end{cases}$$

Hardware Typical hardware suitable for deep learning is a large cluster of power-consuming GPUs constantly doing calculations and in need of cooling. And even when they are idle they consume energy. Spiking Neural Networks only consume energy when a spike is fired due to their event-driven nature. Running an SNN on traditional hardware would not make optimal use of its power-efficient properties, therefore neuromorphic chips have been invented. These chips have several neurons on their board connected by many synapses. The design of these neurons is such that energy is only consumed when a spike fires. Examples of these chips are Loihi [6, 30] and SpiNNaker [14, 28]. A drawback of SNNs is that neuromorphic hardware is not yet easily available so most of the time we have to do with simulations. There are some nice simulators like Nengo[1] and Brian[2], however, these will not lead to the same power efficiency as running on dedicated hardware.

2.2 Conversion

Obtaining trained SNNs can be done in two ways, either by training a model with backpropagation through surrogate gradients [38, 9, 13, 25] or by using a pre-trained model and converting it to an SNN using an ANN-to-SNN [31, 3, 8, 4, 33, 11] conversion method. The first method of training an SNN directly with surrogate gradients is still in early development and takes up time and memory that we are trying to save. The other method of converting ANN to SNN means encoding the activations into spiketrains such that the spiketrains activate the same neurons in the output layer of the SNN as in the ANN. To do this, the weights in the ANN model need to be copied to the weights in the SNN to be used in the IF neurons.

Supported components The first efforts to convert ANN to SNN were done by Perez et al. [31]. Following this research, Cao et al. [4] stated that the conversion of the ReLU activation function to the IF neuron is trivial because of similar functionality. The difficulty is in converting other activation functions like softmax or hyperbolic tangent. In most cases these are ignored in SNN conversion, and the spike rates of the outputs are interpreted as is. Another difficulty is the pooling layer in convolutional neural networks. The properties of max-pooling are not compatible with the spiking properties and therefore mostly avoided by existing research. This is noted by [33], who propose using gates that only let spikes of the maximally spiking neurons pass through. In early works [31, 4] the membrane potential is reset to zero when a spike fires. Rueckauer et al. [33]

[1] https://www.nengo.ai/
[2] https://briansimulator.org/

introduce a soft reset by subtracting the threshold from the membrane potential, therefore retaining the information gained from the last input spike before reaching the threshold.

Research into converting different ANN components is still very active with more recent research showing conversion from ResNet to SNN [34]. Designing spiking-specific variants of popular components like LSTM [26, 2] and transformers [41, 24, 39] is also still an evolving field. So much even that for these last two components no suitable conversion method exists at the moment of writing.

Conversion error In the conversion from ANN to SNN an error is introduced since the SNN can only approximate activations with its spikerates. Rueckauer et al. also give a conversion error per neuron per layer of

$$\varepsilon_i^l = \frac{V_i^l(t)}{TV_{th}}. \tag{1}$$

Where $V_i^l(t)$ is the membrane potential of the i-th neuron in the l-th layer. T the number of timesteps and V_{th} the neurons threshold. This shows that the conversion error depends on the threshold and the number of time steps. Lowering the time steps leads to higher error, and raising them leads to lower error, the membrane threshold has the same effect. This error is per layer and is propagated through all the layers of the network. This formula shows the error emerging from the remaining membrane potential not turned into spikes. This way this specific error is mitigated by raising the threshold V_{th}. However, this also means more time steps since the neurons need more time to spike with a higher threshold.

2.3 Quantization

Quantization is a method from the field of signal processing where an analog signal is discretized. For analog numbers the same technique can be used. A number is typically represented by b-bits. These b-bits can represent 2^b different values. This means that quantizing a vector by lowering the number of bits reduces the precision of the vector, but at the same time lowers the memory usage and computational complexity. Quantization can also reduce the bit depth of a trained ANN. The extreme case is a Binary Neural Network (BNN) [18, 37] where the values of the weights and activations are constrained to $+1$ and -1 or 0 and 1 depending on the use of a signed representation. When converting this kind of network to SNN it is executed with a $2^1 = 2$ or $2^{1-1} = 1$ long spiketrain for signed and unsigned values respectively. A generalization on quantization of neural networks is formulated by Zhou et al. [40] with DoReFa-Net. They innovate by also quantizing gradients, something where earlier research lacked, needing still at least 10-bit numbers for the gradients. DoReFa-Net works with a simple quantization function

$$\text{quantize}_k(x) = \frac{1}{2^k - 1}\text{round}((2^k - 1)x) \tag{2}$$

Where k is the desired number of bits and x is the input. The above equation assumes input to be in the range [0,1]. To achieve this they make sure that the weights are in a range of $[-1, 1]$ with a hyperbolic tangent and then normalized to a range of [0,1] before using the quantization function in Equation 2. For the activations, they do not force the range of the values to be between [0,1]. They assume that the values have passed through an activation function, binding the values to a range of [0,1].

2.4 Quantization in conversion

The length of a spiketrain is, in most cases, determined empirically during testing as a tradeoff between latency and accuracy. Related research, however, describes an upper bound of $2^b - 1$ [15, 32, 17, 23]. This bound is derived by linking the different values a b-bit number can represent to the different spike rates a spike train length T can represent. A spike train without spikes does not give any information. Therefore, it is not counted as a state, which explains the -1 in the bound. As noted by [8] every layer carries over approximation errors from previous layers, each layer should thus have a spiketrain length of $2^b - 1$ to accurately approximate its activation values and not cause any carry-over errors when executed sequentially. This leads to a total latency of $2^b - 1 * L$ for sequential execution of the layers [17]. Guo et al. [15] perform very similar work establishing an equivalence of $T_{SNN} \simeq 2^n_{ANN}$ per layer for quantization. This upper bound of approximation leads to the SNN being increasingly less accurate than the ANN when fewer timesteps are used, following this theory. As mentioned before the number of levels a b-bit deep number can represent is the same as for a spiketrain with $T = 2^b - 1$. For unsigned numbers there are 2^b levels, the upper bound for T becomes $2^b - 1$. So lowering T is similar to quantization to a lower b. This similarity between quantization and SNN spiketrain length can be a good measure of how long the spiketrain should be after conversion.

Short spiketrains are essential for short inference times and low memory usage. As shown in the upper bound a lower bit depth of ANN leads to a shorter spiketrain. To decrease latency there has been work on first quantizing pre-trained models, Post Training Quantization(PTQ), before conversion or training using Quantization-Aware-Training(QAT) to achieve better conversion results with low inference latency. QAT works by simulating quantization during training using clipping operations. Recent research [3, 23] for example shows that QAT leads to very low inference times with the same accuracy as with non-quantized conversion. A large downside of this method is that it requires training a network from scratch using the QAT technique. PTQ is in that regard more effective at saving resources by only quantizing already trained ANNs.

3 Research Question

Conversion from ANN to SNN and quantization are similar in the way they alter the precision of a model's floating-point variables. A quantization algorithm converts a higher bit precision to a lower bit precision, i.e., from a long sequence of ones and zeros to a shorter sequence of ones and zeros. The same happens when an ANN-to-SNN conversion algorithm maps floating-point activations to spike trains. Similar to a floating-point value, the spike train is filled with spikes, denoted as ones and zeros, where the length of the spike train determines the precision of the SNN. The general rule in SNN is that longer spike trains translate to more precise models. However, to save energy and time, it is desirable to have the least number of time steps you can afford. Theory gives us an upper bound on the latency [17] for deep neural networks,

$$Latency = 2^b - 1 \times L$$

where b is the bit depth of the ANN and L is the number of layers. The bound gives a number of timesteps where the model should perform the same as the unconverted variant. Because this bound is dependent on the number of bits and the number of layers of the ANN, it is good practice to lower the bit depth of the ANN to also lower the latency of the SNN.

Because the number of timesteps is given by an upper bound, the optimal number of timesteps is still a guess after conversion. This motivates the search for an average case where the number of bits can be directly translated to timesteps. Such that this number is a more accurate estimate of the minimum number of time steps needed for the desired accuracy. This will avoid choosing a value for timesteps that is too low, where the accuracy is not high enough, and at the same time not too high, where the latency is unnecessarily large for the problem. The length of the spike train does typically not depend only on the bit depth of the ANN, since some problems might be able to be solved with less bit precision. In addition some components need more granularity in their activation values than others making some components perform better or worse for low bit depths. This makes determining the length of a spike train based solely on the bit depth of an ANN a very complex problem. To approach this problem the relation between standard quantization methods and the number of timesteps is investigated in this work. This can be formulated as the following research question: *What is the relation between the bit depth of an artificial neural network and the number of time steps of the spiking neural network when converted?* The expectation is that the accuracy drop of quantization and of having too few timesteps will be related to each other in such a way that the SNN needs more timesteps when the bit depth is higher. The relation entails that each bit depth can be matched to a number or range of timesteps.

4 Method

Quantization and SNN conversion are in principle both forms of quantization. Where normal quantization is a form of spatial quantization and SNN conversion temporal quantization. Since both are forms of quantization they also introduce error with respect to the original data. For quantization the original data is in a vector of values to be quantized. In SNN conversion the original data consists of the activation values of each layer in the ANN. For uniform quantization with a uniform error distribution of $\varepsilon \sim \text{uniform}(-\frac{\Delta}{2}, \frac{\Delta}{2})$, the variance of the quantization error is defined in the field of signal processing [29] as

$$\Delta = \frac{x_{max} - x_{min}}{2^b - 1} \tag{3}$$

$$\varepsilon_{\text{quantization}} = \frac{\Delta^2}{12}. \tag{4}$$

ANN to SNN conversion has the following per neuron conversion error as defined by Rueckauer et al. [33]

$$\varepsilon_i^l = \frac{V_i^l(t)}{T \cdot V_{th}} \tag{5}$$

These formulas both describe the error of a quantization process. Intuition dictates that there should be a relation between these two errors. Because there is no uniform distribution in the SNN conversion error, these two equations can not be compared to determine T. This can also be traced back to Equation 5, where the error depends on the threshold and the membrane potential of the neurons. These, in turn, depend on the task, model structure and conversion algorithm used. This leads to many variables with complex effects on final results.

As stated in the previous section, the theoretical value for T may be a huge overestimation. The lack of a realistic theoretical basis thus motivates an empirical study.

4.1 Materials

4.1.1 Framework

To test the hypothesis, a framework has been developed. The framework consists of a set of different components.

Quantization, conversion, testing, and accuracy matching have all been split into different components. The quantization component quantizes both weights and activation values, however both are optional. When neither is selected the component just returns a copy of the original model.

The part of the framework that defines a converter uses the quantization component and the SNN conversion library from SpikingJelly[3]. Using these it converts a (quantized) Artificial Neural Network to a Spiking Neural Network. The converter is designed to quantize the ANN layer by layer to investigate the impact of quantization of different layers on the conversion. Although the framework offers this conversion method, this research limits itself to a complete quantization, thus leaving layer-by-layer analysis a subject for further research.

To test the performance of both the quantized networks and the SNNs a test function has been designed. This test function reports on the chosen accuracy metrics for a given test set. The main part of the framework takes a pre-trained ANN as input and converts this model to a spiking version and multiple quantized versions with each a different bit depth. The goal of the framework is to analyze the relation between a model's bit depth and the length of the spiketrain of a converted SNN of this model. To analyse the relationship the framework has a part which matches the accuracy curve resulting from quantization to the accuracy curve of the SNN. This method of measuring is explained in more detail in subsection 4.3.

Quantization For quantization in the framework inspiration is taken from DoReFa-Net [40]. They however assume that the input lies in the range of [0,1]. Using scaling to make their method usable for arbitrary input results in the following formula used in the quantization part of the framework:

$$s = \frac{\max |x|}{2^{k-1} - 1} \tag{6}$$

$$\text{quantize}_k(x) = \text{round}(\frac{1}{s}\ x) \cdot s \tag{7}$$

In this formula s is the scale to which the output should be scaled. Since the input models are pre-trained the output of the quantization should have the same scale as the input only with fewer levels, simulating the possibilities of lower bit numbers. The scaling is necessary for proper inference of the quantized model. Which is mathematically equivalent to DoReFa-Net quantization from Equation 2 for signed, arbitrary-range input. In DoReFa-Net the entire model is quantized, which includes the weights and gradients. The framework does not train quantized models. Therefore gradient quantization is omitted. Also when converting to SNN, the weights of the ANN are copied over without any adjustments. This makes quantizing weights unnecessary and would create an unfair comparison between quantized ANN and SNN.

SNN conversion SpikingJelly is used for SNN conversion, they have built the conversion framework set up by Rueckauer et al. [33]. This work utilizes data-based weight normalization from [10]

[3]https://spikingjelly.readthedocs.io/zh-cn/latest/index.html

and improves it by extending to biases. $W^l \rightarrow W^l \frac{\lambda^{l-1}}{\lambda^l}$ and $b^l \rightarrow b^l/\lambda^l$. With weights W^l biases b^l and maximum activation λ^l. Furthermore, they improve by taking into account outliers in the activations by using only the 99th percentile of activations for normalization. They also combine batch-normalization with the preceding convolutional layers by scaling the weights accordingly. This way, the same happens numerically, but there is no need for conversion of the Batch-Normalization layer. In the following equations (8), (9), (10), the batch normalization is fused into the weights of the previous layer.

$$BN(x) = \frac{\gamma}{\sigma}(x - \mu) + \beta \tag{8}$$

$$\tilde{W}^l_{ij} = \frac{\gamma^l_i}{\sigma^l_i} W^l_{ij} \tag{9}$$

$$\tilde{b}^l_i = \frac{\gamma^l_i}{\sigma^l_i}(b^l_i - \mu^l_i) + \beta^l_i \tag{10}$$

Where x is the input to be transformed. μ is the mean, σ the variance. γ and β are learned during training.

The input data to the framework is analog input, the same as in [33]. The alternative used in previous work of generating a Poisson distribution leads to stochasticity in the input. Following the principles of SNN this makes sense but it also makes model performance more random. To fairly compare the performance of different models there should be no randomness in how the data is presented to the models.

4.1.2 Limitations of the framework

Since the framework builds on the conversion library of SpikingJelly it also suffers from the same limitations. This means the framework is only able to convert CNN or feed-forward networks to SNN. There is, however, the possibility of passing your preferred converter to the framework. Quantization does not suffer from these limitations. Therefore inserting a conversion algorithm compatible with more components will also increase the compatibility of the framework. For now this means the framework will only successfully convert ANNs to SNNs that consist only of convolutional and linear components. As activation function only the ReLU unit is accepted. Also models with MaxPooling, AveragePooling and Batch Normalization will convert properly.

Due to the scaling needed in the quantization function in equation 6 the quantized models will not perform with low bit quantization without retraining or fine-tuning.

4.1.3 Models and data

The framework supports models defined using PyTorch and data using the dataloader class from PyTorch. For the experiments three models are used, a 3-layer CNN, AlexNet [20], and VGG16 [35]. These are all convolutional models of varying depth and size. The 3-layer CNN, depicted schematically in Figure 1, further on also referred to as CNN, is a self-defined convolutional neural network with three 2D convolutional layers with ReLU activation followed by MaxPooling. The last layer of the model is a single fully connected feedforward layer for the classification output. The convolutional layer has a 3x3 kernel. AlexNet and VGG16, in Figure 2 and 3 respectively, are left standard except for the output layer, this layer is adjusted for all the models to the number of classes of the task. AlexNet has a total of 8 layers, of which 5 are convolutional layers with kernels

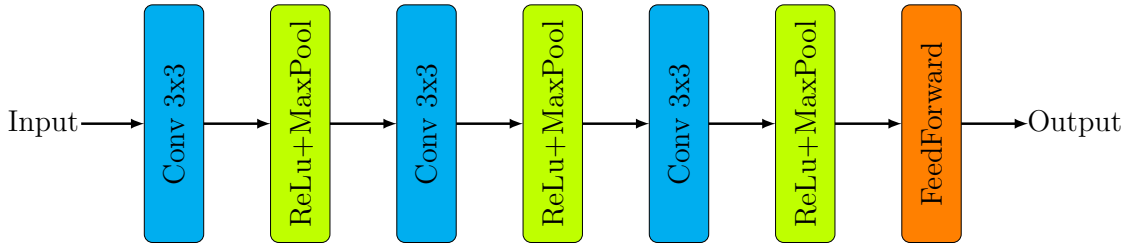


Figure 1: CNN

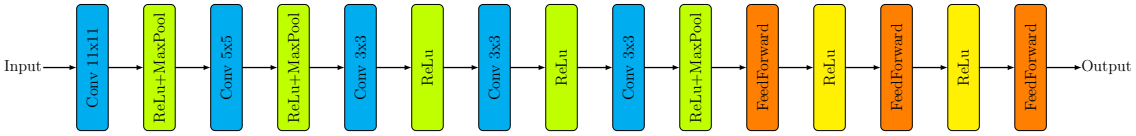


Figure 2: AlexNet

of 11x11 and 5x5. Followed by 3 fully connected feedforward layers. VGG16 has 16 layers in total, of which 13 are convolutional layers with 3x3 kernels. This model also ends with 3 fully connected feedforward layers. Both AlexNet and VGG16 use ReLU activation and MaxPooling, just as in the 3-layer CNN.

MNIST [22], CIFAR-10/100 [19] and ImageNet [7] are the datasets that are used for analysis. MNIST is a handwritten digits dataset containing 60,000 training and 10,000 test grayscale images of 28x28 pixels with 10 classes. CIFAR-10 and CIFAR-100 both consist of 50,000 training and 10,000 test images of 32x32 pixels, where CIFAR-10 has 10 classes and CIFAR-100 has 100 classes. The training images are evenly distributed over the classes. CIFAR-100 is considered harder to learn because it has more classes that are divided into subclasses of similar items. ImageNet, also known as ImageNet (ILSVRC) 2012, has 1000 classes, 1,281,167 training images and 50,000 validation images. The test images are not labelled, so the validation set of ImageNet is used and referred to as the test set from now on. Both the CIFAR datasets and the ImageNet dataset are considered more difficult to learn for a neural network compared to MNIST. MNIST only consists of greyscale images which simplifies the classification process, whereas the other datasets use an RGB color scheme.

For compatibility with all of the models, all images are scaled to 224x224 3-channel RGB images.

4.2 Experimental Setup / Approach

With the use of the quantization-conversion framework described in Section 4.1.1, an analysis is set up. The goal is to analyze the relation between bit depth and time steps. For this to have meaning, first the effect of quantization on different datasets and models has to be examined. Using Equation

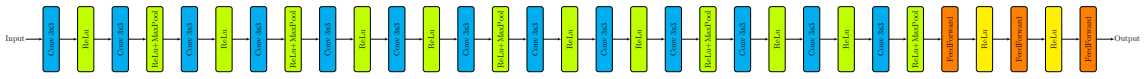


Figure 3: VGG16

2, all of the models are quantized from 32-bit to 16, 8, 7, 6, 5, 4, 3, 2 and 1 bits. This assumes that the significant changes in accuracy happen between bit depths 8 and 1. The accuracy is then measured for all of these bit depths.

As stated before, the theoretical upper bound of the spiketrain length is $2^b - 1$. Previous research has shown that this is a huge overestimation, with spike trains ranging from a few 10s to a few thousand timesteps. This is far of the 4 million timesteps theoretically needed for 32-bit ANN conversion. To investigate how this bound holds in practice, the relation between timesteps and accuracy is analyzed next. For this analysis the conversion framework from [33] implemented in SpikingJelly [12] is used. For each timestep, two accuracies are measured, the accumulated accuracy of the entire spike train integrated over time and the accuracy at the actual timestep. The latter gives insight into the point where the SNN starts to converge on the batch. While the first is the traditional way of interpreting the output of an SNN. If the theoretical bound is indeed a huge overestimation, it would be more useful to have a known average that one could aim for when running a converted SNN. To investigate whether such an average exists and whether or not this can be linked to quantization the following experiment is set up.

To find a relation between bit depth and timesteps, the accuracy for each bit depth of the quantized ANN is compared to the accuracy of the SNN at different timesteps. The idea of this comparison is to see if there is a relation between the number of bits and the number of time steps. The original unquantized model is converted to a spiking neural network. This SNN is then run for 500 timesteps in which the accuracy converges. Each bit depth can then be matched to the timestep with a corresponding drop in accuracy with respect to the maximum achieved accuracy. Performing this comparison for different tasks and different models, the goal is to determine an average estimate of the latency of an arbitrary model on an arbitrary task.

For each model, pre-trained weights from Hugging Face or torchvision will be used if available, if not, then the model is first pre-trained. For testing, a batch size of 32 is used. The results are averaged over all batches.

After reporting on both the accuracy of quantization and SNN timesteps, the equivalence between the accuracy drop of these two models will be measured to investigate the relation between quantization and timesteps. For the tested levels of quantization, the theoretical bounds in Table 1 exist.

Bit depth	Theoretical length
32	4294967296
16	65536
8	256
7	128
6	64
5	32
4	16
3	8
2	4
1	2

Table 1: Theoretical length of a spiketrain

4.3 Measures

The relation between bit depth and time steps is measured in terms of a performance metric depending on the problem. The ANN and SNN are both measured using the same performance metric. In experimentation only classification tasks are used. The metric used is thus the classification accuracy denoted by the percentage of correctly classified samples. For converted SNNs, both the accuracy of the single timestep and the accuracy of the output of all the previous timesteps combined are plotted. This is also known as the accuracy of the spike train and the most common measure for SNNs. To measure the relation between bit depth and spike train length, a relative accuracy measure is used.

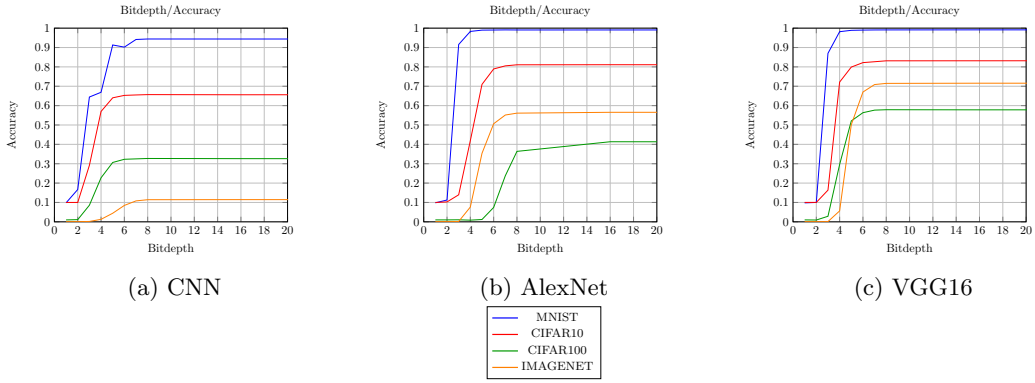
$$\text{Relative accuracy}_{ANN} = \frac{Acc_{bitdepth}}{\max(Acc)} \tag{11}$$

$$\text{Relative accuracy}_{SNN} = \frac{Acc_{timesteps}}{\max(Acc)} \tag{12}$$

5 Experimental Results

5.1 Quantization

To analyze the effect of quantization on the three models, the accuracy is plotted against the bit depth in Figure 4. This is the result of quantization of pre-trained CNNs without any fine-tuning before or after quantization.



(a) CNN (b) AlexNet (c) VGG16

Figure 4: Comparison of the effect of quantization on different datasets. The graphs show absolute accuracy at each bitdepth.

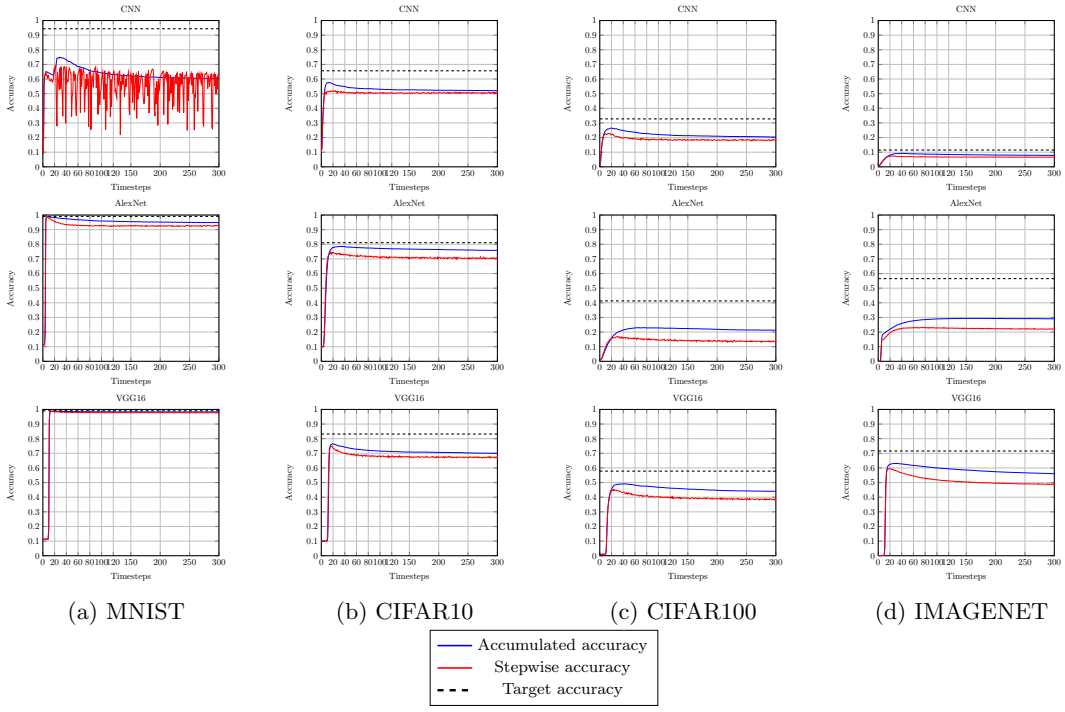
The results show an increase in accuracy when the bit depth increases. For the 3-layer CNN in Figure 4a there is a dip in accuracy at 6 bits for the MNIST dataset. This differs from all other results of the other models and datasets, where the accuracy drops constantly with each bit drop. In general the accuracy drops faster for more complicated models. The exceptions are with 3-layer CNN on MNIST and for AlexNet and VGG16 on CIFAR-100.

Overall the models show better performance for low bits when the dataset is less complicated. Also the accuracy drop seems steeper when the performance of the model is better. Only AlexNet

shows a drop in accuracy when quantizing to 8 bits on CIFAR-100, the rest of the models maintain their accuracy to 8 bits or even lower.

5.2 SNN

When a spiking neural network is presented with data, the membrane potential starts at 0. Then it takes some time before neurons start firing. This process is made visible with the *stepwise accuracy* lines in Figure 5. We observe that this line is a lot less stable than the *accumulated accuracy*, representing the accuracy of the class predicted by the accumulated timesteps. Also the maximum accuracy achieved by the *accumulated accuracy* is higher. For CNN on MNIST, the stepwise accuracy is especially unstable.



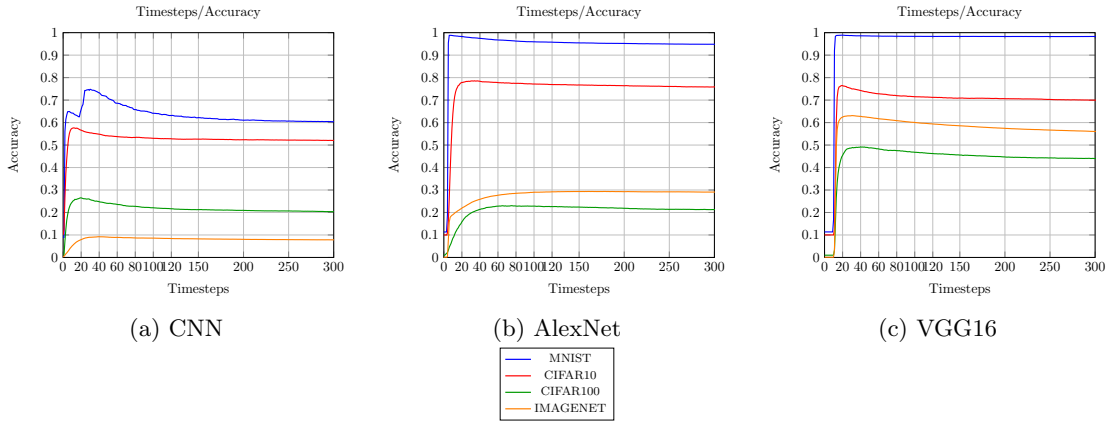
(a) MNIST (b) CIFAR10 (c) CIFAR100 (d) IMAGENET

Figure 5: Absolute SNN accuracy for CNN, AlexNet and VGG on MNIST, CIFAR-10, CIFAR-100 and ImageNet. The graphs show a comparison between the absolute stepwise and absolute accumulated accuracy for each combination of model and dataset. See section 6 for further explanation.

For all models the accuracy starts at random and increases over time to reach a peak. After the peak, performance drops a little and converges at a lower point. For AlexNet the peak for top accuracy is not as different as for the other models. For the CNN on MNIST there is also a dip in performance just before reaching its peak accuracy. The shape is somewhat the same as for the dip in accuracy noticed with quantization.

For most of the model-dataset combinations, the performance of the SNN is not level with the performance of the ANN. Only for the deeper models is this target reached, although this is limited

to the MNIST dataset. VGG16's SNN remains close to the targeted accuracy for all of the datasets, staying within a 10% error margin.



(a) CNN (b) AlexNet (c) VGG16

Figure 6: Absolute accumulated SNN accuracy at different timesteps for CNN, AlexNet and VGG16 on MNIST, CIFAR-10, CIFAR-100 and ImageNet.

In Figure 6 the accumulated accuracy of the different SNNs is shown. Here the four datasets are compared to each other in terms of their performance. For CNN, the performance is shown to decrease with the complexity of the data. AlexNet and VGG16 seem to struggle with CIFAR-100 compared to ImageNet. Both show lower accuracy on CIFAR-100 than on ImageNet. The gap to CIFAR-10 is significant for the smaller models, but for VGG16 this gap is relatively small. VGG16 does need more time to have accurate predictions as opposed to CNN, which has reasonable results almost from the start. AlexNet fits in between. For all the models, it can be seen that the peak shifts to the right when the dataset gets more difficult.

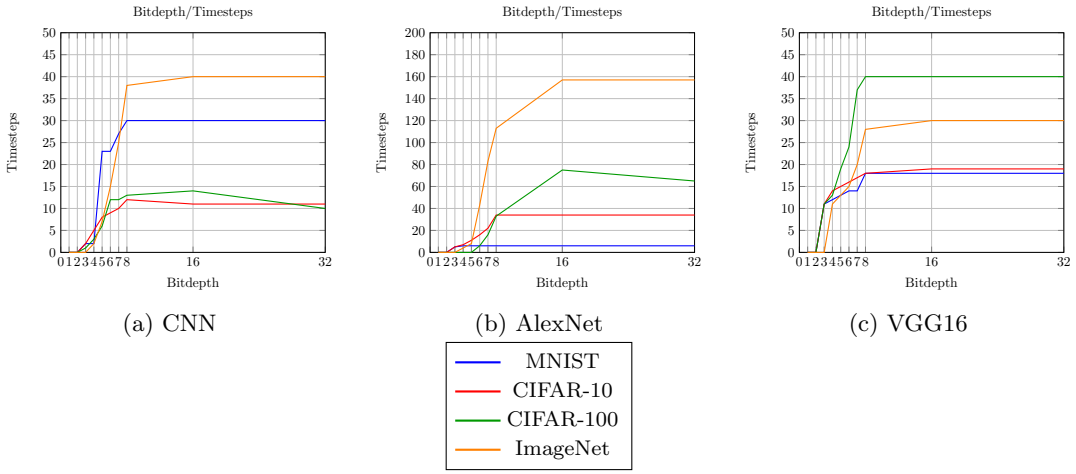
5.3 Relation

For the relationship between bit depth and spike train length, the relative accuracy to the previous peaks is measured following the measurements in Equation 11 and 12. The SNN's accuracy is also measured relative to the target accuracy. Where the target accuracy is the accuracy of the unquantized 32-bit ANN. The results of this comparison are shown in Table 3, 4 and 5.

In Figure 7 the bit depth is linked to the number of timesteps for every dataset for each model. It shows the number of time steps needed for approximately the same accuracy drop as with quantization, the tables in the Appendix show which accuracy drops are matched. This gives insight into how the different datasets impact the relationship between bits and timesteps. Also the impact of different models on the same datasets is shown in Figure 8. This gives a comparison of the impact of using different models.

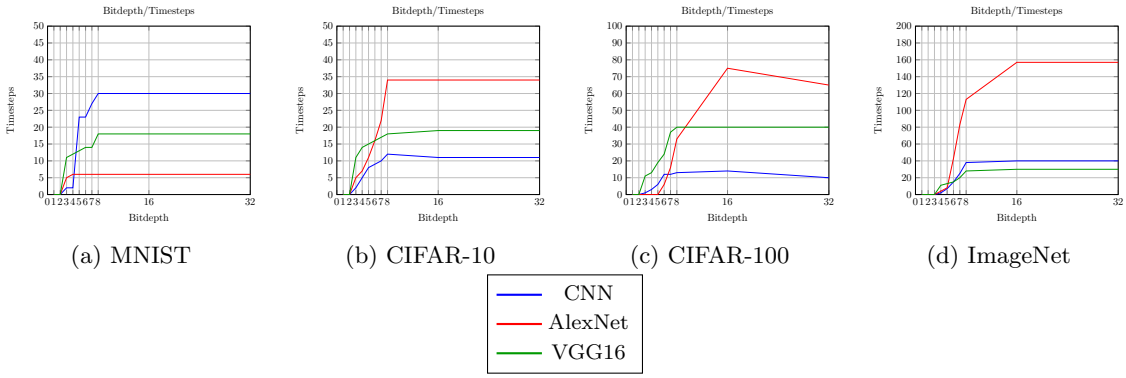
In Figure 7 is shown that for AlexNet and VGG16, the models need more timesteps for the more complex datasets and fewer for the easier datasets. CNN shows deviant behavior, needing more timesteps for MNIST than the more complex CIFAR datasets. When the timesteps do not increase anymore with higher bit depths, it means the model does not need more timesteps to achieve the same accuracy drop as quantization.

In Figure 8 Alexnet needs the most timesteps for all datasets except for MNIST where it needs the least. Also for CIFAR-100 AlexNet shows a lower timestep to have the same accuracy drop as with 8 bits. VGG16 is for almost all of the models in between in terms of needed timesteps. Only for ImageNet does VGG need the fewest timesteps.



(a) CNN (b) AlexNet (c) VGG16

Figure 7: Relation between quantization bit depth and SNN time steps for different models



(a) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) ImageNet

Figure 8: Relation between quantization bit depth and SNN time steps for different datasets

6 Discussion

6.1 Quantization

The quantization results in Figure 4 show an increase in performance when the dataset gets easier. Only for AlexNet and VGG the hardest dataset does not have the lowest accuracy. This might be because for AlexNet and VGG networks pretrained on ImageNet are used. These parameters

available in the PyTorch torchvision library are known for their state-of-the-art performance. For CIFAR-100, CIFAR-10 and MNIST, such pretrained parameters are not readily available. Probably because not all tricks for state-of-the-art performance were used while training AlexNet and VGG on CIFAR-10, CIFAR-100 and MNIST the performance of CIFAR-100 will not go beyond that of ImageNet. In theory this should not be the case and CIFAR-100 should outperform ImageNet. For VGG it can be seen that the bit depth where the accuracy starts to drop is shifted more to the left for CIFAR-100, indicating that VGG needs less information for CIFAR-100 images to correctly classify them. AlexNet being a smaller network does need more information from the bits even though the problem is easier. For the custom build CNN no pretrained parameters were available of course. Therefore all of the datasets have been trained on in the same manner, resulting in the expected placement of the curves. Only for MNIST there is a weird kink visible in Figure 6 where the model improves for a lower bit depth. This probably has to do with the nature of the MNIST dataset which consists of simple grayscale images. The rounding that happens in quantization might round a grey pixel on an edge the right way making the image less noisy after the first layer. The simplicity of MNIST provides space for this, more complex datasets would have more complex pixels where this effect would not appear. The other models also do not show this effect due to their higher complexity which makes them more robust to this kind of rounding anomalies.

6.2 SNN accuracy

Following the results of the previous section it is observed that the spiking neural networks peak in performance before plateauing at a lower level than the peak. This might be explained by the firing rates of the internal neurons. The membrane potential $V_i^l(t)$ is set to 0 at the start of execution. The first neurons to fire are those with high frequencies. After them the lower frequency nodes start firing. There are probably more low-frequency neurons than high-frequency ones. Due to their lower frequency, their spikes contribute less often to the output. When the low frequencies are also diverse in their frequencies, the low-frequency neurons all start to fire at different moments. Each neuron has information encoded in its firing rate, although some nodes might have suffered a conversion error and produce some noise. So in the first timesteps, only the high-frequency nodes fire their output, leading to the lower starting accuracy, and then more and more neurons will start to fire until the most frequent frequency. This explains why the stepwise accuracy does not exceed a certain value and the accumulated accuracy converges to a point lower than the peak as seen in Figure 5. As to why the accuracy dips after the peak and stabilizes at a lower accuracy, this also has to do with oversaturation of the spike trains. When the low-frequency output nodes also start to fire, too many spikes fire at the same time for the wrong classes, making the SNN less certain about its prediction in the long run. This has the consequence that the accuracy stabilizes at a lower level.

6.2.1 Stepwise accuracy

For CNN on MNIST again an anomaly is observed. The single-timestep accuracy is very variable compared to the single-timestep accuracy of the other datasets and models. This means that the CNN on MNIST is very susceptible to noise. Since the low accuracy timesteps only occur after more than 20 timesteps it is probably caused by the low frequency neurons causing noise when they start to fire after 20 timesteps. With MNIST some classes are quite similar. In cases where the CNN is in doubt which of the two classes is the correct where it under normal circumstances it would choose the right one it can be easily misled when looking at the single timestep accuracy.

What is curious to see is that the performance taken at a single timestep reaches almost the same height as the accumulated performance, which uses all information from the earlier timesteps. This could mean that the SNN does not necessarily need the information gained from the earlier time steps to make accurate predictions. Meaning that for inference on an SNN, the first couple of timesteps should not be measured since the internal state of the nodes is not yet adapted to the input they are given. This could also mean that a single timestep is enough to make a reasonably accurate prediction after discarding the first few timesteps. Following this, one could argue that the usual method of integrating over timesteps is not per se needed, especially when looking for fast inference. This insight is one of the more important observations of this work.

6.2.2 Overestimation in theory

All of the models have been trained using 32-bit floating-point decimals. During the inference it is shown that 16 bits and in most cases even 8 bits are enough for level performance with the unquantized 32-bit ANN. In Figure 7, it is clear that the upper bound of $2^b - 1$ timesteps is an overestimate. Given that the models need at least 8-bit floats, all of the SNN models reach their limit in terms of accuracy before the theoretical time step limit of $2^8 = 265$ is reached. Some of the models even show a significant increase in accuracy before 20 timesteps. This is possible because a spiking model might not need all timesteps to work accurately for a dataset. This means the theory of $T = 2^b - 1$ dictates that the model can be quantized to $b = \log_2(T + 1)$ bits where T is the number of timesteps needed for the desired accuracy. This value for T can be read from the tables in the appendix or Figure 7. This is a useful finding when wanting to know to how far the bit depth of a model can be theoretically reduced before losing accuracy.

6.3 Relation

Looking at the relational graph in Figure 7 AlexNet needs the most timesteps to approximate the bit depths accuracy for most of the datasets. This means that for these datasets AlexNet is less efficient in conversion. AlexNet has 5 convolutional layers where the first layer has a 11x11 kernel and the second a 5x5 kernel. This makes AlexNet aware of more surrounding pixels compared to CNN and VGG, which use only 3x3 kernels. Because of this AlexNet needs more time before the SNN converges. This also explains why the curves for AlexNet in Figure 6 are more gradual and have less of a peak than the other two models have.

6.3.1 Formalization of the relation

Judging from the relationship curves in Figure 7 and 8 the best fit for the relation is a logistic function.

$$\frac{L}{1 + e^{-k(x-x_0)}} \tag{13}$$

Where L determines the height of the curve, k the steepness and x_0 determines the midpoint of the curve. A higher value for L means that the SNN needs a lot of timesteps to achieve the best performance it can. The midpoint of the curve in combination with the steepness tells how well the SNN can keep up with the quantized ANN in terms of relative error. The curve depicts where the relative quantization error is equal to the relative conversion error. When the curve has plateaued it means that increasing the number of bits no longer needs more timesteps to have equal relative error. At this point the ANN and SNN do not improve anymore. On the steep part of the curve

the SNN needs more timesteps to reach the same relative performance as the quantized ANN. This means that at this point the ANN improves its performance fast. When the curve is steep it means that the SNN is less capable of keeping up with the quantized ANN. However the SNN can still reach the same relative error as the quantized ANN. Intuitively more difficult datasets will result in higher values for L. This is not necessarily seen in the results in Table 2. Therefore it is hard to make conclusions about the appropriate parameters for each dataset and model.

Fitting the logistic curve to the data in Figure 8 and 7 results in the parameter values in Table 2.

Dataset	**Model**	L	x_0	k
MNIST	Conv	29.2060	4.7283	1.6950
	AlexNet	6.0645	2.7580	2.9648
	VGG	15.7468	3.0697	1.5017
CIFAR-10	Conv	11.1355	4.2891	1.1386
	AlexNet	35.5318	6.0122	0.8231
	VGG	17.1081	3.0278	1.8934
CIFAR-100	Conv	13.5143	4.9928	1.3899
	AlexNet	74.7281	8.1436	1.2214
	VGG	41.6989	5.1216	0.8059
ImageNet	Conv	40.9026	6.4469	1.2321
	AlexNet	155.3011	6.9809	1.1463
	VGG	30.3218	5.6700	0.7426

Table 2: Estimated parameters (L, x_0, and k) for each dataset and model.

If we analyze the relation between bit depth and time steps from a dataset point of view the time steps corresponding to bit depths can be read in Figure 8. This gives for MNIST for 3-layer CNN 23 steps for 5 bits for reasonable performance AlexNet is still reasonable with 3 bits and this corresponds to 5 steps. This already shows how different the results are for different models on the same dataset. AlexNet is almost twice as large as the CNN but the needed timesteps at the same bit depth are not nearly twice as large. This rules out a linear relation between bit depth, time steps, and the number of layers. Looking at the size of the datasets there is a doubling in timesteps for AlexNet and VGG16 going from CIFAR-10 to CIFAR-100. These datasets are comparable since they come from the same set of data and only differ in the number of classes. AlexNet also needs twice as many bits to reach the peak performance in quantization. This corresponds to the doubling of the time steps. Suggesting a linear relation between the number of classes of a dataset of the same complexity. This means that increasing the classes by a 10 fold the number of timesteps to reach maximum performance is approximately doubled. This is translated in the relation in Equation 13 with L. For AlexNet this trend continues to ImageNet where the number of timesteps double again when moving to 1000 classes. Formalizing this trend into the equation would not result in correct results for the other models.

The equation thus gives a formalization of the relation seen in the relational curves. However, ideally, one would want to use the formula to determine the number of timesteps needed for the same relative error as a quantized ANN, given the bit depth. With this formula the function parameters need to be filled in but the current results do not give the needed information to fill in all of these values correctly for any unseen convolutional model and dataset.

7 Conclusion and Further Research

In conclusion, the relation between bit depth and SNN timesteps found in theory is more or less correct. However the actual relation rises not as steep as described by the theoretical relation. Also the lowest possible bit depth needs to be used. Meaning the bit depth at which there is no or negligible loss in performance. Furthermore we found that the first timesteps do not give more information but rather slow down the inference and should therefore be discarded until descent performance is reported in a single timestep.

There is not enough information to determine an exact relation between quantization and SNN timesteps. Although we found that the bit depth of an ANN can be used as a heuristic for the number of timesteps when the lowest performing bit depth is taken into account. In addition, more insight into what has an influence on the relation is gained. For quantization there are no dynamic properties, and the accuracy drop is based on the model, dataset and initial accuracy. SNN accuracy depends on the same variables and more. The method of conversion for ANN-to-SNN probably has a large impact on the resulting SNN performance. For this, more in-depth research into different conversion methods is needed.

The aim of this research was to investigate the relation between bit depth and time steps. It has been made clear that both quantization and SNN inference roughly follow the same shape regarding accuracy. The steepness and plateauing level of the curve depend, for both, on the depth of the model and the complexity of the dataset.

This research is limited to convolutional models at the moment. Due to the difference in architecture of recent models, it is interesting to expand on the research of ANN to SNN conversion to other components frequently used in artificial neural networks, like transformers and recurrent neural networks. Seeing the difference in conversion performance and relation for the different models on the same datasets leads us to believe that the components used in an ANN have a significant impact. Therefore, when there are more conversion possibilities, it is also good to reproduce this research on these converted models to prevent overestimating the number of time steps needed.

References

[1] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[2] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.

[3] T. Bu, W. Fang, J. Ding, P. Dai, Z. Yu, and T. Huang. Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. *arXiv preprint arXiv:2303.04347*, 2023.

[4] Y. Cao, Y. Chen, and D. Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 2015.

[5] G. Datta, Z. Liu, J. Diffenderfer, B. Kailkhura, and P. A. Beerel. Timesteps meet bits: Low-latency, accurate, & energy-efficient spiking neural networks with ann-to-snn conversion. 2024.

[6] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[8] S. Deng and S. Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. *arXiv preprint arXiv:2103.00476*, 2021.

[9] S. Deng, Y. Li, S. Zhang, and S. Gu. Temporal efficient training of spiking neural network via gradient re-weighting. *arXiv preprint arXiv:2202.11946*, 2022.

[10] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. ieee, 2015.

[11] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[12] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, and Y. Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023.

[13] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34:21056–21069, 2021.

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[15] M. Guo, Q. Li, J. Cheng, L. Chen, et al. Qac: Quantization-aware conversion for mixed-timestep spiking neural networks. *International Conference on Learning Representations*, 2025.

[16] S. Hisaharo, Y. Nishimura, and A. Takahashi. Optimizing llm inference clusters for enhanced performance and energy efficiency. *Authorea Preprints*, 2024.

[17] Y. Hu, Q. Zheng, X. Jiang, and G. Pan. Fast-snn: Fast spiking neural network by converting quantized ann. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12):14546–14562, 2023.

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[19] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[21] L. Lapicque. Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *Journal de Physiologie et de Pathologie Générale*, 9:620–635, 1907.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[23] C. Li, L. Ma, and S. Furber. Quantization framework for fast spiking neural networks. *Frontiers in Neuroscience*, 16:918793, 2022.

[24] Y. Li, Y. Lei, and X. Yang. Spikeformer: a novel architecture for training high-performance low-latency spiking neural network. *arXiv preprint arXiv:2211.10686*, 2022.

[25] S. Lian, J. Shen, Q. Liu, Z. Wang, R. Yan, and H. Tang. Learnable surrogate gradient for direct training spiking neural networks. In *IJCAI*, pages 3002–3010, 2023.

[26] A. Lotfi Rezaabad and S. Vishwanath. Long short-term memory spiking networks and their applications. In *International Conference on Neuromorphic Systems 2020*, pages 1–9, 2020.

[27] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[28] C. Mayr, S. Hoeppner, and S. Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018*, pages 277–280. IOS Press, 2019.

[29] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Pearson, Upper Saddle River, NJ, 3rd edition, 2010. See Section 3.8, "Quantization and Quantization Noise", pp. 216–224.

[30] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer, and M. Davies. Efficient neuromorphic signal processing with loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 254–259. IEEE, 2021.

[31] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2706–2719, 2013.

[32] P. Ramesh and G. Srinivasan. Pascal: Precise and efficient ann-snn conversion using spike accumulation and adaptive layerwise activation. *arXiv preprint arXiv:2505.01730*, 2025.

[33] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[34] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrellas. Towards greener llms: Bringing energy-efficiency to the forefront of llm inference. *arXiv preprint arXiv:2403.20306*, 2024.

[37] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 65–74, 2017.

[38] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[39] M. Yao, J. Hu, Z. Zhou, L. Yuan, Y. Tian, B. Xu, and G. Li. Spike-driven transformer. *Advances in neural information processing systems*, 36, 2024.

[40] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[41] Z. Zhou, Y. Zhu, C. He, Y. Wang, S. Yan, Y. Tian, and L. Yuan. Spikformer: When spiking neural network meets transformer. *arXiv preprint arXiv:2209.15425*, 2022.

8 Appendix

Convolutional model	Bitdepth	Rel. Acc.	Accumulated	Rel. Acc.	Rel. Acc. to actual	Timestep	Rel. Acc.	Rel. Acc. to actual
MNIST	32	1.000	30	1.000	0.792	21	1.000	0.732
	16	1.000	30	1.000	0.792	21	1.000	0.732
	8	1.000	30	1.000	0.792	21	1.000	0.732
	7	0.997	27	0.997	0.790	21	1.000	0.732
	6	0.956	23	0.951	0.753	19	0.972	0.711
	5	0.967	23	0.951	0.753	19	0.972	0.711
	4	0.709	2	0.750	0.594	2	0.807	0.590
	3	0.683	2	0.750	0.594	2	0.807	0.590
	2	0.177	0	0.119	0.095	0	0.129	0.095
	1	0.104	0	0.119	0.095	0	0.129	0.095
CIFAR-10	32	0.999	11	0.999	0.877	17	0.999	0.791
	16	0.999	11	0.999	0.877	17	0.999	0.791
	8	1.000	12	1.000	0.877	21	1.000	0.792
	7	0.997	10	0.997	0.874	17	0.999	0.791
	6	0.994	9	0.992	0.870	16	0.994	0.788
	5	0.975	8	0.980	0.860	9	0.979	0.776
	4	0.867	5	0.852	0.747	4	0.829	0.656
	3	0.444	2	0.460	0.403	2	0.506	0.401
	2	0.152	0	0.173	0.152	0	0.192	0.152
	1	0.152	0	0.173	0.152	0	0.192	0.152
CIFAR-100	32	0.999	10	0.998	0.735	13	1.000	0.655
	16	1.000	14	1.000	0.737	13	1.000	0.655
	8	0.997	13	0.997	0.734	13	1.000	0.655
	7	0.994	12	0.993	0.732	13	1.000	0.655
	6	0.993	12	0.993	0.732	13	1.000	0.655
	5	0.937	6	0.923	0.680	6	0.931	0.610
	4	0.681	3	0.717	0.529	3	0.764	0.500
	3	0.250	1	0.169	0.125	1	0.188	0.123
	2	0.031	0	0.044	0.033	0	0.050	0.033
	1	0.031	0	0.044	0.033	0	0.050	0.033
ImageNet	32	1.000	40	1.000	0.804	27	1.000	0.646
	16	1.000	40	1.000	0.804	27	1.000	0.646
	8	0.996	38	0.996	0.801	23	0.995	0.643
	7	0.943	25	0.943	0.759	18	0.954	0.616
	6	0.739	15	0.752	0.605	12	0.764	0.494
	5	0.386	7	0.395	0.318	6	0.371	0.240
	4	0.113	2	0.111	0.089	2	0.129	0.083
	3	0.016	0	0.020	0.016	0	0.024	0.016
	2	0.009	0	0.020	0.016	0	0.024	0.016
	1	0.009	0	0.020	0.016	0	0.024	0.016

Table 3: The relation between bitdepth and timesteps in VGG16 for ImageNet-1k, CIFAR-10 and CIFAR-100

AlexNet	Bitdepth	Rel. Acc.	Accumulated	Rel. Acc.	Rel. Acc. to actual	Timestep	Rel. Acc.	Rel. Acc. to actual
MNIST	32	1.000	6	1.000	0.997	6	1.000	0.997
	16	1.000	6	1.000	0.997	6	1.000	0.997
	8	1.000	6	1.000	0.997	6	1.000	0.997
	7	1.000	6	1.000	0.997	6	1.000	0.997
	6	0.999	6	1.000	0.997	6	1.000	0.997
	5	0.999	6	1.000	0.997	6	1.000	0.997
	4	0.992	6	1.000	0.997	6	1.000	0.997
	3	0.924	5	0.966	0.963	5	0.980	0.977
	2	0.114	0	0.115	0.114	0	0.115	0.114
	1	0.099	0	0.115	0.114	0	0.115	0.114
CIFAR-10	32	1.000	34	1.000	0.969	16	1.000	0.918
	16	1.000	34	1.000	0.969	16	1.000	0.918
	8	1.000	34	1.000	0.969	16	1.000	0.918
	7	0.993	22	0.993	0.962	15	0.991	0.909
	6	0.973	16	0.973	0.943	12	0.974	0.894
	5	0.875	11	0.872	0.844	9	0.858	0.787
	4	0.517	7	0.483	0.468	7	0.607	0.557
	3	0.172	5	0.203	0.197	4	0.152	0.139
	2	0.127	0	0.127	0.123	0	0.134	0.123
	1	0.123	0	0.127	0.123	0	0.134	0.123
CIFAR-100	32	0.999	65	0.998	0.556	30	1.000	0.415
	16	1.000	75	1.000	0.557	30	1.000	0.415
	8	0.880	33	0.883	0.492	18	0.875	0.363
	7	0.575	16	0.573	0.319	10	0.595	0.247
	6	0.180	6	0.197	0.109	4	0.157	0.065
	5	0.029	0	0.043	0.024	0	0.058	0.024
	4	0.021	0	0.043	0.024	0	0.058	0.024
	3	0.025	0	0.043	0.024	0	0.058	0.024
	2	0.024	0	0.043	0.024	0	0.058	0.024
	1	0.024	0	0.043	0.024	0	0.058	0.024
ImageNet	32	1.000	157	1.000	0.519	76	1.000	0.410
	16	1.000	157	1.000	0.519	76	1.000	0.410
	8	0.992	113	0.992	0.515	75	0.992	0.407
	7	0.975	83	0.975	0.506	45	0.976	0.400
	6	0.895	43	0.894	0.464	25	0.894	0.366
	5	0.625	8	0.626	0.325	6	0.629	0.258
	4	0.135	4	0.012	0.006	4	0.017	0.007
	3	0.003	0	0.003	0.002	0	0.004	0.002
	2	0.002	0	0.003	0.002	0	0.004	0.002
	1	0.002	0	0.003	0.002	0	0.004	0.002

Table 4: The relation between bitdepth and timesteps in AlexNet for ImageNet-1k, CIFAR-10 and CIFAR-100

VGG16	Bitdepth	Rel. Acc.	Accumulated	Rel. Acc.	Rel. Acc. to actual	Timestep	Rel. Acc.	Rel. Acc. to actual
MNIST	32	1.000	18	1.000	0.997	14	1.000	0.996
	16	1.000	18	1.000	0.997	14	1.000	0.996
	8	1.000	18	1.000	0.997	14	1.000	0.996
	7	0.999	14	0.999	0.997	13	0.999	0.995
	6	0.999	14	0.999	0.997	13	0.999	0.995
	5	0.998	13	0.998	0.996	13	0.999	0.995
	4	0.991	12	0.994	0.992	12	0.995	0.991
	3	0.878	11	0.932	0.930	11	0.938	0.934
	2	0.101	0	0.115	0.115	0	0.115	0.115
	1	0.099	0	0.115	0.115	0	0.115	0.115
CIFAR-10	32	1.000	19	1.000	0.919	18	1.000	0.900
	16	1.000	19	1.000	0.919	18	1.000	0.900
	8	0.999	18	0.998	0.917	18	1.000	0.900
	7	0.994	17	0.994	0.914	15	0.994	0.895
	6	0.988	16	0.991	0.911	15	0.994	0.895
	5	0.961	15	0.974	0.896	14	0.979	0.881
	4	0.867	14	0.930	0.855	13	0.909	0.818
	3	0.197	11	0.162	0.149	11	0.242	0.218
	2	0.121	0	0.131	0.120	0	0.134	0.120
	1	0.120	0	0.131	0.120	0	0.134	0.120
CIFAR-100	32	0.999	40	1.000	0.850	23	1.000	0.782
	16	0.999	40	1.000	0.850	23	1.000	0.782
	8	1.000	40	1.000	0.850	23	1.000	0.782
	7	0.996	37	0.996	0.846	23	1.000	0.782
	6	0.974	24	0.975	0.829	19	0.973	0.760
	5	0.900	19	0.909	0.772	17	0.907	0.709
	4	0.512	13	0.537	0.456	13	0.581	0.454
	3	0.049	11	0.066	0.056	10	0.024	0.019
	2	0.016	0	0.020	0.017	0	0.022	0.017
	1	0.017	0	0.020	0.017	0	0.022	0.017
ImageNet	32	1.000	30	1.000	0.881	20	1.000	0.832
	16	1.000	30	1.000	0.881	20	1.000	0.832
	8	0.999	28	0.999	0.880	18	0.998	0.830
	7	0.990	20	0.990	0.872	16	0.990	0.823
	6	0.936	15	0.944	0.832	14	0.945	0.786
	5	0.700	13	0.770	0.678	13	0.830	0.690
	4	0.079	11	0.055	0.048	11	0.075	0.062
	3	0.002	0	0.002	0.001	0	0.002	0.001
	2	0.001	0	0.002	0.001	0	0.002	0.001
	1	0.001	0	0.002	0.001	0	0.002	0.001

Table 5: The relation between bitdepth and timesteps in VGG16 for ImageNet-1k, CIFAR-10 and CIFAR-100