Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

EVENT-BASED OBJECT RECOGNITION USING ANALOG AND SPIKING NEURAL NETWORKS

Semesterthesis

Nicolas Känzig
D-ITET

Supervisor: Yulia Sandamirskaya
Advisor: Bodo Rueckauer
Profesor: Tobi Delbruck

Institute of Neuroinformatics
University of Zurich and ETH Zurich

Abstract

In this project object recognition in the context of a robotics predator/prey navigation scenario using analog and spiking neural networks is performed. A dataset containing event-based data acquired by a dynamic vision sensor (DVS) is used to train a convolutional neural network (CNN) using frames obtained by accumulation of the DVS event streams. After that the network is converted into a spiking CNN representation and accuracy is evaluated using synthetically generated input spiketrains as well as real event-based DVS input. The reasons for the loss in accuracy that occurs in the SNN after conversion were analyzed. While very little loss in accuracy arised when driving the spiking network with synthetic spikes, for DVS input a bigger gap in accuracy was measured. The conversion of the biases as well as the sparse and non-uniform nature of the DVS event streams were identified to be the main reasons for the observed loss after conversion. In a last step a simple multilayer perceptron architecture was implemented on the neuromorphic platform spiNNaker, evaluating performance and the feasibility of using the platform for performing direct SNN training using backpropagation inspired algorithms instead of ANN-SNN conversion.

Contents

1 Introduction

Object recognition is a fundamental problem in the study of computer vision and is paramount to many applications in robotics. The open issue is always to determine how features of an image should be extracted and represented. In traditional computer vision these visual features are usually generated using frame-based data. Frame-based cameras capture scenes at fixed time periods independent of the dynamics of the underlying scene. This often results in acquisition of huge amounts of redundant data as most pixels will not change from one frame to the next [10]. Also the typically low frame-rate strongly affects the latency of the system which is a major drawback in real-time applications in robotics where reaction time can be crucial.

One approach to counter these problems is to acquire the data in an event-driven fashion using biologically inspired vision sensors such as DVS [13], ATIS [23] or DAVIS [2]. Instead of capturing static images of the scene these sensors record pixel intensity changes with high temporal precision thus leading to both low latency and low bandwidth as events are only triggered if a change occurs in the observed scene. However, as the data now is composed by a sequence of events, traditional frame-based computer-vision algorithms are not applicable anymore.

1.1 Related Work

Various advances in artificial visual sensing using event-based data have already been made. An obvious way to process event based data is to create frames by accumulating events over fixed time intervals or by using constant event counts. Being in the frame domain again, conventional approaches such as convolutional neural networks (CNN) can thus be applied for feature extraction and object classification [18], [1], [14]. However, this naive approach ignores key advantages of event based sensors, in particular their low data rate and the high temporal precision. Also much of the energy efficiency is lost if the event stream is interpreted by conventional synchronous processors. These synchronous systems face a latency-power tradeoff where low latencies can only be achieved by the use of high sample rates, leading to increasing power consumption. Using asynchronous architectures for processing the event-based data will be more cost efficient.

Spiking neural network (SNN) architectures inspired by the brain have been identified due to their asynchronous nature as a tool that could be utilized in this task.

On one hand a number of studies have shown that SNNs can be successfully constructed by converting conventionally trained analog neural networks (ANNs) such as CNNs [25], [6]. One major issue that is reported is that while these neural networks seem to work well using synthetic data generated artificially from frame images (like MNIST), where the gray level of an image pixel is mathematically transformed into a stream of spikes using, switching to real input data captured with neuromorphic vision sensors such as DVS often leads to significant loss in accuracy.
On the other hand recent work has successfully explored direct training in the spiking domain using backpropagation inspired techniques for directly training multi-layer SNN architectures [12], [19], [20]. Using this techniques SNNs can be trained directly on real event-based data recorded by neuromorphic sensors and therefore it is claimed that the inherent timing statistics of the produced spiking signals can better be captured than with ANN-conversion based techniques [12].

Using both methods, classification accuracies similar to those obtained in ANNs can be achieved, potentially running on a fraction of the energy budget when using dedicated neuromorphic hardware. However, just as hardware acceleration through GPUs has played a fundamental role in the advancements of deep-learning, there is an increasing need for powerful neuromorphic hardware that enables efficient computation of event-based SNN training and inference.

1.2 Objective and outline

In this project object recognition in the context of a robotics predator/prey navigation scenario is performed. The dataset from [18] is used to train and evaluate several neural network architectures, where the purpose of the trained networks is to steer a predator robot in the direction of a prey robot.

This thesis compares conventional deep-learning ANN architectures such as CNNs and multilayer perceptron (MLP) networks with their purely event-based SNN counterparts.
In a first step, two conventional ANN architectures are trained and evaluated on the task at hand using frame-based data only. The proposed CNN architecture from [18] is replicated and accuracy is compared to a simple multi-layer perceptron (MLP) network with only one hidden layer.
In a second step the trained ANN architectures are converted into their SNN counterparts. The functionality of the resulting SNN architectures is analyzed and accuracy is evaluated using both synthetic and non-synthetic input spike data.
Finally after simulation a converted spiking MLP architecture is imple-

mented on the spiNNaker [7] neuromorphic hardware platform.

2 Deep-learning approaches

As a starting point two standard frame-based deep-learning approaches (CNN, MLP) have been implemented, trained and evaluated. The dataset from [18] was used to replicate the results of the paper and the proposed CNN architecture was compared to a MLP network architecture.

The simplest possible implementation of the predator needs to recognize in which of the three vertical regions of its field of view the prey is located (Left/Center/Right and Non-visible), leading to a standard classification problem with four different classes (L, C, R, N).

2.1 Dataset

The dataset from [18] consists of twenty recordings with a total duration of 1.25 hours from a Dynamic and Active Pixel Sensor (DAVIS) [2]. The DAVIS camera was mounted on the predator robot, recording different scenes where the prey robot is visible or not visible. The recordings thus contain both conventional image frames (APS) as well as event-based data (DVS). The DVS sensor data was acquired using the AER protocol [4] and is stored using the AEDAT 2.0 fileformat [8]. Each event has a timestamp and the address of the corresponding pixel assigned. All recordings are labeled with the position (pixel [x,y]) of the prey robot. The needed 4-class labels (L, C, R, N) can easily be extracted using this information.

In total the dataset contains 75452 APS frames and 256983 artificially generated DVS frames containing 5000 events each were obtained.
The processed dataset was separated into train- and testset where the test fraction accounts for 20% of the samples. As the dataset consists of video recordings where consecutive frames are highly correlated in time, it is important to shuffle the training data, such as not to obtain entire minibatches of highly correlated samples during training. However, the data must be shuffled after splitting the dataset as otherwise train- and testset would contain highly correlated data and thus evaluation on the test set would not be meaningful. The data was stored using HDF5 files.

2.2 Preprocessing

The DAVIS camera records with a resolution of 240x180. During preprocessing of the dataset the data was subsampled to 36x36 which has found to

be the minimum size by which the robot can still be recognized by human eye [18]. The APS frames are subsampled using nearest-neighbor interpolation. Also in [18] the APS dataset was expanded by creating falsely over- and under-exposed APS data by shifting the gray values of the frames by a fixed amount and by clipping the data out of range. This increases the total size of the dataset and balances out the APS/DVS frame ratio. The processing of the DVS data is described in the following section.

2.2.1 Generating frames from event-based data

Whereas the APS frames can be labeled and fed directly into the ANNs after subsampling as they are, this is not the case for the event-based DVS recordings. A way to use the DVS data is to create artificial frames from the recorded event streams. There are two different methods to create these frames from the event-based data:

(1) Accumulate events with constant event count

(2) Accumulate events within fixed time periods

In [18] and in this work method (1) with an event count of 5000 was used. This way the frame rate is proportional to the rate of change of the scene and it is guaranteed that each frame contains valuable information, whereas method (2) would lead to very sparse and noisy frames in time intervals where not much contrast changes happen in the recorded scene and thus the DVS sensor wouldn't trigger many events.
As the DVS events also have a polarity (ON/OFF) there are several ways to perform the accumulation operation. Two methods are listed in the following:

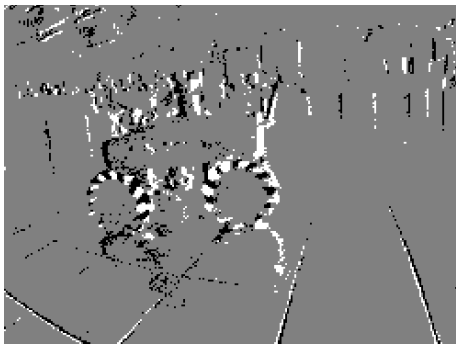
(A) Don't distinguish between event polarities
- Pixel intensity is incremented by δ for each event (ON & OFF)

(B) Use positive sign for ON events and negative sign for OFF events
- Increment/decrement by δ for ON/OFF events. Negative event counts are possible.

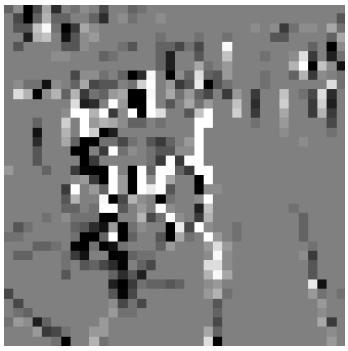
After accumulation, the event counts are always scaled to $[0, 1]$ resulting in greyscale image frames. After scaling, to remove outliers the DVS histograms are clipped at three times their standard deviation σ computed around the mean μ of the originally acquired DVS histogram.

In [18] the frames are initialized with 0.5 pixel intensities and method (1B) with $\delta = 0.005$ is used for the event accumulation. During the scaling operation the histogram level which corresponds to zero events is held at 0.5 to

avoid unwanted flickering which would complicate the recognition task [18]. This makes sense as the network will eventually learn that 0.5 pixel values correspond to zero event counts and are thus not relevant for the classification task. However, this comes at the cost of distorting the original frame histogram. Fig. 1 shows an example of the resulting DVS frames.



(a) 240x180, frametype (1B), zero event pixels hold at 0.5 as in [18]



(b) 36x36

(c) 240x180, frametype (1A)

(d) 36x36

Figure 1: DVS frames, full resolution and subsampled versions

2.3 Convolutional neural networks (CNN)

Convolutional neural networks [11] describe an artificial neural network topology that is inspired by the biological visual cortex and is mostly used in computer vision tasks. CNNs are composed of several layers of processing neurons, where each neuron in a layer is connected to a neighborhood of neurons in the next layer. A CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. The input layer of a CNN is always convolutional and its parameters consist of a set of learnable filters which will extract meaningful features from the visual

input. Pooling layers combine the outputs of neuron clusters at one layer into a single neuron in the next layer leading to a dimensionality reduction. Different pooling operations have been proposed in literature. The common max pooling uses the maximum value from each of a cluster of neurons at the prior layer while average pooling uses the average value. The convolutional and pooling layers in CNNs combined thus extract meaningful visual features with low dimensionality from the input images. At the output fully connected layers are then driven with the extracted image features and a softmax layer can be used to produce outputs for classification tasks.

2.3.1 Implementation

In this work the architecture proposed in [18] was implemented. The CNN consists of the following layers:

1. 36x36 Convolutional layer - 4 feature maps (5x5) - ReLu
2. Max pooling layer - poolsize=2x2, stride=2
3. 18x18 Convolutional layer - 4 feature maps (5x5) - ReLu
4. Max pooling layer - poolsize=2x2, stride=2
5. Fully connected layer 40 Neurons - ReLu
6. Softmax layer - 4 Neurons

Neurons: 5884
Synapses: 242592
Trainable parameters: 6472

Note that due to the weight sharing property of CNNs the number of trainable weights is much lower than the number of synapses. Fig. 2 illustrates the implemented CNN architecture. The network was implemented in python using Keras with a Theano backend. The DVS frames for training were generated using the same method as in [18], outlined in section 2.2.1. Training was performed in batches of size 32 minimizing crossentropy-loss using an Adadelta optimizer. The training set is shuffled after each epoch. An epoch of training took 15min on average on a i7-4770 CPU and after ten epochs the model reached its maximum performance.

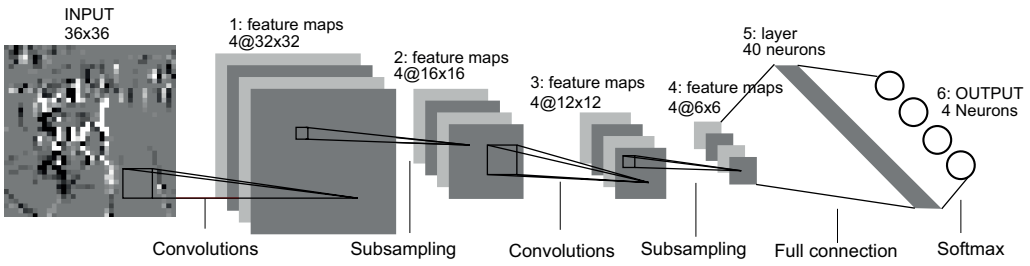


Figure 2: Implemented CNN architecture (Grafic modified from [11])

2.3.2 Results

Three different models were trained, one using only DVS data, one only using APS frames and the third was trained with the complete combined dataset. [18] states that training separate CNNs for DVS and APS invariably resulted in overfitting. To increase the amount of training data they found that training a single CNN driven using both datasets combined resulted in higher overall accuracy. This result could not be replicated in this work. Fig. 3 shows the measured test accuracies and crossentropy-losses during 30 epochs of training for the three different models. It can be observed, that combining the DVS and APS frames into one training set did not yield higher accuracy and also the models do not seem to be particularly prone to overfitting as the test loss hardly increases. Table 1 lists the measured accuracies of the three models after 10 epochs of training using both the DVS and APS testsets. Combining the data sets even decreased accuracies slightly compared to the models trained with the seperate datasets. Another interesting measure is the accuracy of the DVS-model fed with APS data and the accuracy of the APS-model driven with DVS-data. While the DVS-model still performs clearly better than random, the APS-model fails completely when classifying DVS frames.

Despite the efforts of creating DVS frames with an appearance as similar as possible to static APS frames, the resulting histograms still differ significantly which complicates the recognition task when combining the two datasets. However, in cases where the separate datasets really are too small to train models without overfitting, increasing the size of the dataset by combining DVS and APS frames might be beneficial. As this is not the case with the dataset at hand and the implemented model, and as event-based data is of particular interest for this work, in the following only models trained with the DVS dataset will be used. Also it has been found that DVS models trained with frametype (1A) perform slightly better ($\sim$2%) than models trained with frametype (1B) as used in [18]. Furthermore, there are other strong reasons as outlined in section 3.3.3 for the use of this

frametype, having the goal of converting the system into a SNN architecture in mind.

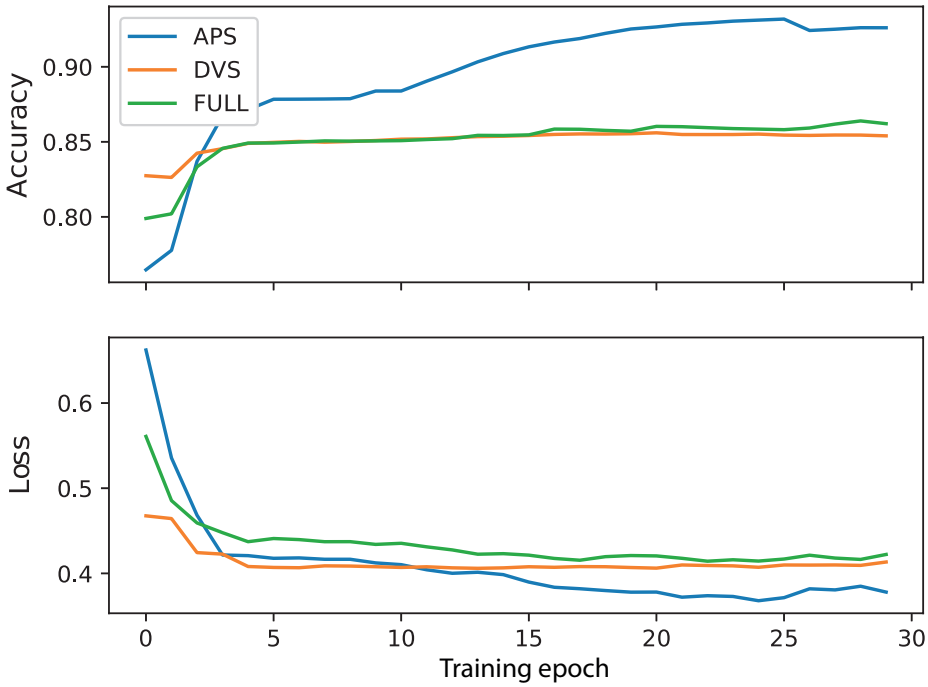


Figure 3: Test accuracy and crossentropy-loss of the models trained with DVS, APS and the combined dataset over 30 epochs. The curves were smoothed using a second order moving average filter.

Model	**DVS Accuracy** **N L C R**	**APS Accuracy** **N L C R**
DVS	0.85 (0.82) 0.90 0.80 0.84 0.72	0.71 (0.68) 0.89 0.5 0.47 0.47
APS	0.56 (0.32) 0.99 0.05 0.16 0.05	0.92 (0.86) 0.97 0.72 0.86 0.89
Full	0.84 (0.77) 0.93 0.70 0.76 0.68	0.86 (0.78) 0.98 0.66 0.73 0.74

Table 1: Measured accuracies using DVS and APS testsets. The first two accuracy measures in a row entry state the total accuracy and the total accuracy averaged over classes (in brackets) on the corresponding testset, whereas the four measures below state the class accuracies.

2.4 Multi-layer perceptron (MLP)

The multi-layer perceptron describes one of the most common feedforward artificial neural network architectures in deep-learning. Despite their simplicity these models perform reasonably well on a variety of learning tasks. MLPs usually consist of an input layer, one ore more fully connected hidden layers and an output layer. Fig. 4 illustrates such an architecture with one hidden layer.

MLP neural networks can be used to extract features from image data, however, generally it is not practical to apply this architecture to images especially when working with high resolution datasets. A very high number of neurons would be necessary, even in an architecture with only one hidden layer due to the full connectivity. Furthermore, by transforming the 2-dimensional image data into 1-dimensional vectors to fit the dimension of the input layer, valuable information content of the images gets discarded.

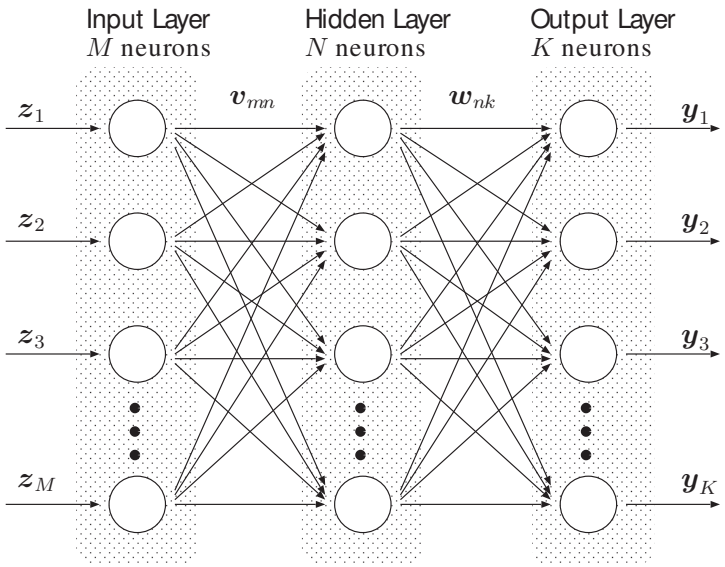


Figure 4: MLP architecture (Illustration from [9])

2.4.1 Implementation

In this work a simple MLP architecture with only one hidden layer was implemented. As in many applications the use of two or more hidden layers did yield better accuracy.

Whereas the number of neurons in the input and output layers are determined by the dimension of the input images and the number of classes of the

classification task, the number of neurons in the hidden layer is a tunable parameter. Three networks with 64, 32 and 16 hidden layer neurons were trained using the DVS data where the frames were generated using method (1A) from section 2.2.1. As in the last section, again batches of size 32 and an Adadelta optimizer to minimize cross-entropy loss was used during training.

Table 2 lists the number of neurons and synapses for the different configurations. As can be observed, due to the fully connected architecture the number of synapses grows rapidly when increasing the number of hidden neurons.

	64	**32**	**16**	**8**
Neurons	1364	1332	1316	1308
Synapses	83200	41600	20800	10400
Trainable parameters	83268	41636	20820	10412

Table 2: Number of neurons and synapses for architectures with 64, 32, 16 and 8 hidden layer neurons.

2.4.2 Results

Despite the higher complexity in terms of trainable parameters of the trained MLP networks they don't perform as well as the CNN architecture proposed in section 2.3.1. This result was expected, as CNNs in contrast to MLPs are tailored for deep vision tasks. Still, the MLP models achieved decent accuracy.

Fig. 5 shows the measured test accuracy and crossentropy-loss during 10 epochs of training for the four different models. It can be seen that all models already after three epochs of training start to overfit as the test-loss begins to increase at this point. The architecture is therefore much more prone to overfitting than the CNN architecture analyzed in the previous section, which was expected as complexity in terms of trainable parameters is significantly higher.

Table 3 lists the measured accuracies of the four models after ten epochs. It can be observed that despite the tremendous decrease in the number of trainable parameters when going from 64 to 8 hidden layer neurons, the measured gap in accuracy is at only 3%. It can be deduced that the required model complexity to solve the classification task at hand is fairly small, which matches the observations in section 2.3.2 where a somewhat minimalistic CNN already achieved decent results.

In the experiments in the following sections MLP architectures with 16 hidden layer neurons will be used, as this configuration yields a fair trade-off between accuracy and computational costs.

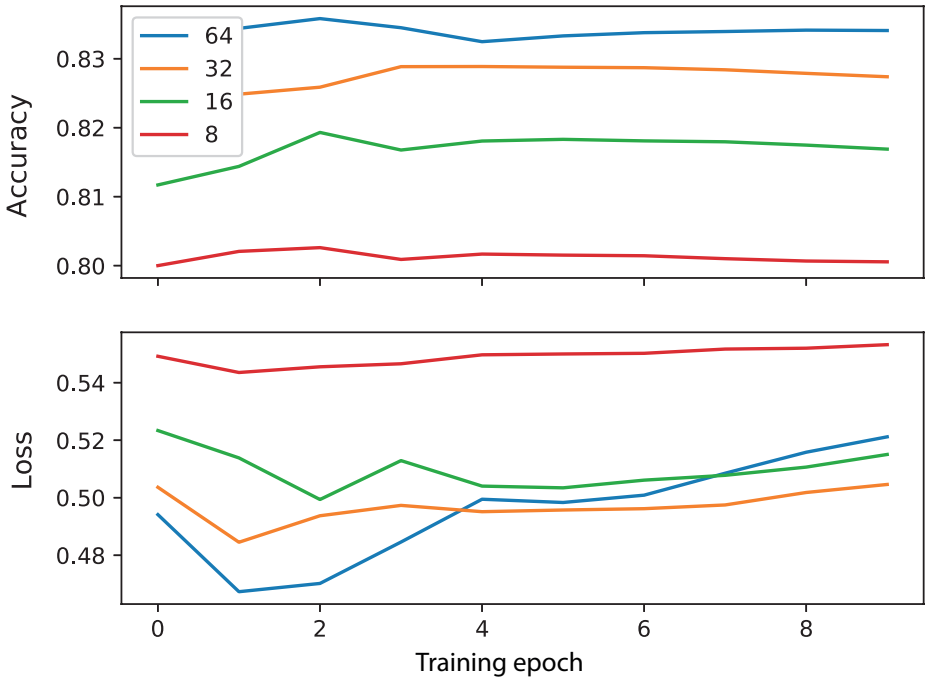


Figure 5: Test accuracy and crossentropy-loss of the trained models with 64, 32, 16 and 8 hidden neurons over 10 epochs of training. The curves were smoothed using a second order moving average filter.

3 Event-based approach with spiking neural networks

3.1 Spiking neural networks

The neuron model deployed in spiking neural networks (SNN) is strongly inspired by the functionality of biological neurons. Unlike conventional neuron models, spiking neurons do not fire at each propagation cycle but only when their membrane potential crosses a firing threshold. After firing, the membrane potential drops and the neuron produces a spike that is propagated to all connected neurons. Incoming spikes will increase or decrease the membrane potentials of the corresponding neurons proportionally to the weights of the connecting synapses.

Model	DVS Accuracy N L C R
64	0.83 (0.76) 0.91 0.72 0.78 0.65
32	0.82 (0.76) 0.90 0.73 0.77 0.63
16	0.81 (0.74) 0.91 0.70 0.75 0.61
8	0.80 (0.74) 0.78 0.72 0.76 0.61

Table 3: Measured accuracies of the MLP models with 64, 32, 16 and 8 hidden layer neurons. The first two accuracy measures in a row entry state the total accuracy and the total accuracy averaged over classes (in brackets) on the corresponding testset, whereas the four measures below state the class accuracies

Theory has shown that SNNs are at least as computationally powerful as conventional neurol models being used in deep-learning [15]. The asynchronous event-based nature of SNNs comes with several advantages. While regular ANNs process information in a synchronized manner, meaning that every neuron in a layer is evaluated before the information can progress to the next layer, in SNNs neurons are only activated, when addressed by an event [16]. It has been shown that by the use of dedicated event-based hardware, power consumption and latency can be reduced by serveral orders of magnitude. In [17] an architecture consuming about 1000 times less energy than conventional synchronous architectures was presented.

3.2 SNN Training

Training SNNs is difficult as due to their non-differentiable nature gradient-descent based methods can not be applied directly. Furthermore, "back-propagation rules such as used in deep-learning often rely on the immediate availability of network-wide information stored with high-precision memory during learning, and precise operations that are difficult to realize in neuromorphic hardware" [20].

Although recently several back-propagation inspired algorithms for SNNs, achieving accuracies similar to those obtained in conventional ANNs, have been proposed, there is an increasing need for dedicated neuromorphic hardware that supports SNN training.

On the other hand it has been shown that traditional ANNs can be converted into SNNs with almost zero loss in accuracy [25]. In other words one can obtain a trained SNN by first training an ANN and then converting it to a SNN representation. Using the trained ANN models from the last section, this approach will be evaluated in the following.

3.3 ANN-SNN Conversion

The main idea of ANN-SNN conversion is that the firing-rates of the resulting spiking architecture should be closely correlated to the analog activations in the original ANN. This correlation can be achieved by directly mapping the weights of the ANN to the SNN model - in other words using the same weights for the SNN as for the ANN. The architecture of the converted SNN stays largely the same while a spiking neuron model is deployed. However, some layers such as max-pooling or softmax can not directly be converted into an equivalent spiking representation by just changing the neuron model. [25] shows how these operations can be approximated using spiking architectures.

However, by simply mapping the weights directly usually a significant loss in accuracy is observed after conversion. There are several possible reasons for this loss:

Low firing-rates
: The resulting firing-rates in the converted SNN might be too low and neurons thus won't receive sufficient input to cross their firing-threshold. This problem could be overcome by introducing normalization layers into the network to boost ANN activations, increasing the resulting firing-rates in the converted SNN. In case that activations in the ANN are generally low, the weights of the model can be normalized to increase the activation values and thus the resulting spiking-rates [5].

Saturating firing-rates
: Firing-rates in time-stepped simulations or in dedicated event-based hardware are usually limited to a finite value. If activation outputs in the ANN are too big, the firing-rates in the converted SNN will reach their maximum value and thus saturate which will inevitably decrease accuracy. To overcome this problem weight and biases can be normalized such as to avoid saturation of the spiking neurons [5].

Non-uniform spiketrains
: Depending on the distribution of the spike input of the SNN, it can

happen that a set of spikes over- or underactivate a specific feature set due to non-uniformity of the resulting spiketrains [5].

Biases

Representing biases in SNNs in a meaningful way is not straightforward. Some conversion methods therefore exclude biases during training of the ANN models to be converted. However, this restriction often comes with losses in accuracy of the ANN already before conversion. In [25] the biases are converted into constant input currents proportional to the bias values, flowing into the corresponding neurons. The neurons integrate this constant currents leading to a constant increase of the membrane potential in each timestep.

Softmax layer

In ANNs the output stage usually consists of a softmax layer producing strictly positive outputs that can be interpreted as probabilities in classification tasks. It is not straightforward to convert this mechanism into a spiking representation. What mostly has been done is to predict the class that corresponds to the output neuron which produced the most spikes while feeding an input sample into the network. However, this approach fails when neurons in the final layer receive too many negative inputs leading to very negative membrane potentials. In this case the output neurons will hardly produce any spikes [25].

[21] proposed a spiking softmax stage, where output spikes are triggered by an external Poisson generator with variable firing rate. The spiking neurons do not fire on their own but simply accumulate their inputs. When the external generator determines that a spike should be produced, a softmax competition according to the accumulated membrane potentials is performed [25].

Max-pooling layers In analog CNNs most often max-pooling layers are used for down-sampling the feature maps. However, computing maxima with spiking neurons is non-trivial. Some conversion methods restrict the ANN models to the use average-pooling layers which can directly be represented in the spiking framework. However, this often results in lower accuracy of the ANN.

[25] proposes a simple mechanism for spiking max-pooling, in which output units contain gating functions, which only let spikes from the maximally firing neuron pass.

Negative activations

Mapping negative activations to firing-rates is difficult as there are no negative spikes. This is a problem if the neurons in the ANN use an activation function that can produce negative outputs such as $tanh()$. However, models using ReLU activation have strictly positive outputs and are thus not affected.

3.3.1 Spiking neural network conversion toolbox

The methods proposed in [25] describe the current state-of-the-art in ANN-SNN conversion. The conversion method provides spiking implementations of common CNN operations such as softmax, max-pooling and batch-normalization and in contrast to previous works enables the conversion of ANN networks with non-zero bias. Losses below 1% in accuracy are reported after conversion.

An open-source implementation is available at [24], and was used in the following. The toolbox also comes with an integrated simulator which was used for all simulations performed in this section. The simulator uses a simple integrate-and-fire (IF) spiking neuron model [3].

3.3.2 Input spiketrains

Several methods can be used to generate the spiketrains to be used as input for the SNNs. Due to the lack of truly event-based datasets acquired with neuromorphic vision sensors, in recent work the spiketrains were often generated synthetically from frame-based image datasets. The most common method is to use poisson spike generators and drive their firing-rate with the intensity of the corresponding input pixels. However, the stochastic nature of the generated poisson spiketrains introduces noise into the network. This can be circumvented by using analog input values in the input stage and compute with spikes from there on. The analog pixel values are converted into currents flowing into the neurons of the input layer where the currents are integrated to membrane potentials [28], [25].

The predator/prey dataset used in this work contains real event-based data recorded with a DVS sensor. The DVS events can be directly interpreted as input spikes. when neglecting their polarity.
Recent work [26] has reported that while converted SNNs seem to work well using synthetic input data generated from frame images, using real event-based data as input often leads to significant drop in accuracy. In this work simulations with both synthetically generated and original event-based input were performed. The outcomes are discussed in the results section below.

3.3.3 Selecting the frametype for ANN training

As outlined in section 2.2.1 there are several possible methods for generating static image frames from event-based recordings. However, not all frametypes are suitable if the final goal is to convert the trained ANN into a SNN model.
The frames used in [18] are not ideal for this purpose for the following reasons. Pixels corresponding to zero event counts or zero ON-OFF sums have 0.5 values, so when generating synthetic input spikes using poisson generators or analog inputs as explained in the last section, the 0.5 pixel values will thus be converted into spiketrains or non-zero analog input currents. However, it is not meaningful to apply an input to the SNN, when actually in the original data there is none. This strongly contradicts fundamental principles of event-based computing, where the idea is perform computations only if events occur in the system. Furthermore, while the converted system will work using synthetically generated input spiketrains, it will not when feeding the original event-based data into the network. In the SNN the DVS events can be interpreted as spikes and can thus be fed directly into the network, so the input holds no information on the artificially induced non-zero input spiketrains corresponding to pixel addresses with zero event counts. One could preprocess the input-spikes by adding non-zero spiking ground-rates for the 0.5 pixels, then the question remains how to distinguish the ON and OFF events during spike generation, as negative spikes do not exist in our SNN model.

A much simpler and more elegant solution is using the frame generation method (1A) as explained in section 2.2.1, where frames are initialized with zero pixel values and where pixel intensities are incremented for each event while not distinguishing between ON and OFF events. This way pixel intensities with value zero clearly correspond to zero-event counts, and in the SNN no input will be generated. For these reasons in the following only models trained with this frametype will be evaluated.

3.3.4 Weight and bias normalization

In previous work weight normalization was introduced to tackle the problem of saturating or low firing-rates. While in SNNs the maximum firing-rate is restricted to a maximum value given by the inverse time resolution of the simulator or the hardware being used, there is no such restriction for the activations in ANNs. [5] proposed a data-based normalization scheme where weights are simply scaled using the maximum activation values observed

while feeding the training set or a smaller subset of the data through the network. However, [25] found that while this approach ensures that the firing rates never saturate, the drawback is that if the dataset contains outlier samples that lead to very high activations, the weights will be scaled using these outlier activation values, resulting in low firing-rates for the majority of the remaining samples. [25] expands this normalization concept to systems with non-zero biases and instead of using the maximum observed activation of all samples, a less conservative approach is taken by scaling the weights and biases using the p-th percentile of the activation distribution, where values between 99 and 100 for p are reported to perform particularly well. The weights and biases are thus scaled as follows where l denotes the layer and λ_l the p-th percentile activation:

$$\tilde{W}_l \frac{\lambda_{l-1}}{\lambda_l} = W \qquad \tilde{b} = b \frac{1}{\lambda_l} \tag{1}$$

3.4 Results

For the evaluation of the converted SNN a new smaller test set containing 100 samples per class was uniformly sampled from the complete DVS testset as used for reporting accuracies in section 2.3.2. Using the complete testset was not practical as runtimes of the SNN simulations are orders of magnitude higher that when performing ANN predictions in the Keras/Theano framework.

3.4.1 Converting the original CNN model

In a first step the CNN model trained with DVS frames as described in section 2.3.1 was converted without any modifications. Using analog synthetic input, saturation of the spike-rates in all layers of the spiking CNN was observed. The measured accuracy on the testset was with 74.5% slightly lower than ANN accuracy (80.05%). By normalizing the weights and biases using the 99.9-th percentile activation as described in section 3.3.4, SNN accuracy increased to 78.25%, almost closing the gap to ANN. Fig. 6 shows the correlations between activations and firing-rates in the second convolutional layer before and after normalization, where the saturation of the spiking-rates without normalization can be clearly observed.

While the model worked well for synthetic input it initially failed completely when using DVS input spiketrains. It was observed that the firing-rates of the DVS inputs were much lower compared to the resulting spiketrains using synthetic input. Thus the biases had a much stronger impact when using

DVS data, so the constant currents induced by the biases were too dominant with respect to the currents induced by the DVS input spiketrains. Comparing the spiketrains using DVS input with the resulting spiketrains when applying no input, lead to the conclusion that the vast majority of the triggered spikes were induced only by the bias and not by the actual input of the SNN (Fig. 8).

With respect to this outcome it is no surprise that the model in this case completely failed to perform correct predictions. The four biases in the output layer were identified as the main issue, as the bias value of the output neuron corresponding class N was significantly higher then the other biases. For this reason the model mostly predicted class N irrespective of the input sample class.

To reduce the impact of the bias currents, the bias values were scaled down by a constant factor. This scaling factor has been determined experimentally where a value of 150 has found to perform reasonably well, achieving 68.5% accuracy on the test set. Potentially the results could be further improved by using different scaling factors for each layer adapted to the corresponding bias values. Fig. 7 shows correlation plots of the first convolutional layer before and after rescaling the biases. It can be seen that the rescaling operation increases the correlation between activations and spikerates dramatically.

As an alternative approach to rescaling the bias currents of the converted network one could train the ANN using bias regularizers to keep the bias values small. However, this adds an additional constraint to the ANN during training and increases complexity of the hyperparameter tuning of the model as appropriate values for the regularizer parameters have to be determined.

Table 4 summarizes the achieved accuracies on the test set using synthetic input and the original DVS spiketrains.

	Accuracy	**Accuracy loss**
ANN	80.05%	-
SNN (analog)	74.5%	-5.55%
SNN (analog), p=99.9	78.25%	-1.8%
SNN (DVS), bias/150	68.5%	-11.5%

Table 4: Accuracies for the converted SNN models using analog input with and without weight normalization with the 99.9 percentile activation and DVS input with scaled biases.

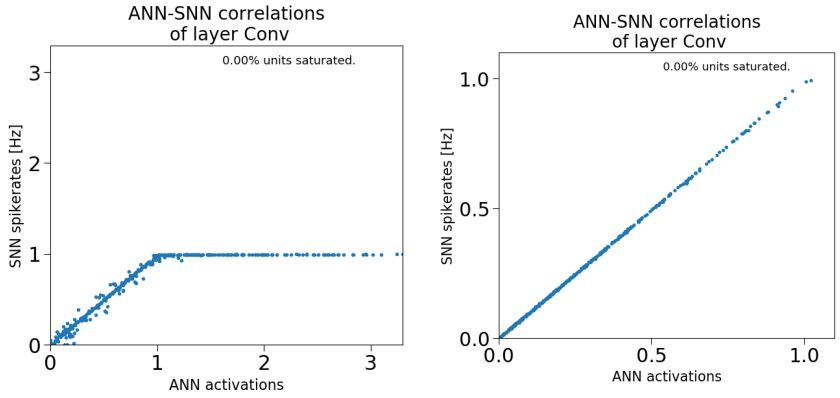


Figure 6: Correlations between activations and firing-rates in the second convolutional layer before (left) and after (right) normalization with the 99.9-th percentile activation using analog input. Activation values > 1 lead to saturation of the spiking-rates.

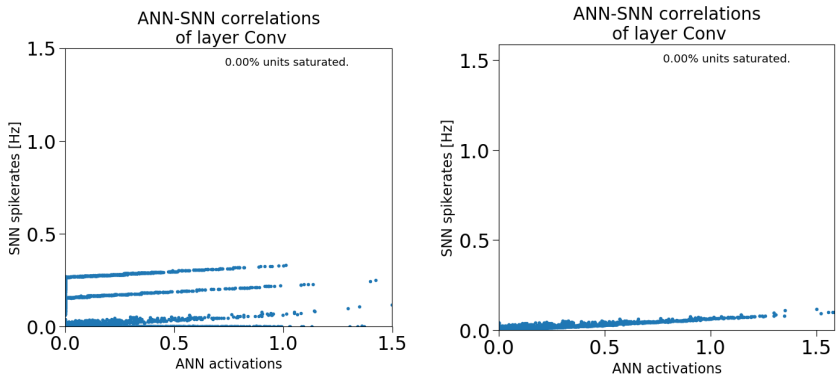


Figure 7: Correlations between activations and firing-rates in the first convolutional layer for DVS input before (left) and after (right) downscaling the biases by a factor 150.

3.4.2 Zero-bias model

As found in the last section converting ANN bias values into constant currents flowing into the spiking neurons can be critical when sparse input spiketrains are applied to the network. Even so, after reducing the bias currents a performance gap between the ANN and SNN driven by DVS input remains.

To identify the other reasons for this discrepancy, a new ANN model with a zero-bias constraint was trained and again converted into a SNN. So the analysis of the remaining issues could be conducted without being interfered by the effect of the biases on the losses during the conversion.

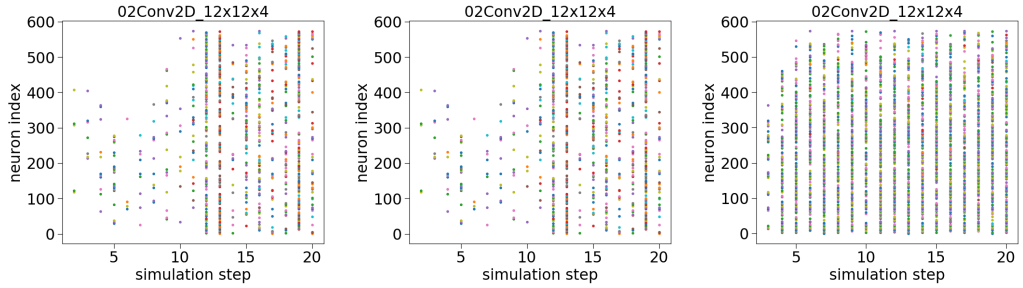


Figure 8: Spiketrains in the second convolutional layer using no input (left), DVS input (middle) and analog input (right) during the first 20 simulation steps. The spiketrains in the first two figures look almost identical, so it can be deduced that most spikes are induced by the bias currents when applying DVS input.

ANN accuracy of the zero-bias model was at 81.25% and thus even increased lightly with respect to the previous model. The converted SNN model achieved an accuracy of 80.50% for analog input and 71% with DVS input, which is slightly higher than the results achieved in the last section using a model with scaled biases. However, this gap potentially could be closed by further optimizing the bias scaling factors for all layers. Again the achieved accuracies are summarized in table 5.

	Accuracy	**Accuracy loss**
ANN	81.25%	-
SNN (analog), p=99.9	80.5%	-0.75%
SNN (DVS)	71%	-10.5%
SNN (DVS), single subsampling	75.1%	-6.15%

Table 5: Accuracies for the converted SNN models using analog input with weight normalization and DVS input with standard subsampling and single event subsampling, where only one event of a subsampled patch is kept.

Subsampling

One of the remaining reasons for the observed performance gap using DVS input data has found to be the subsampling mechanism being used during generation of the spike trains. The resolution is subsampled from 240x180 to 36x36 where the subsampled pixel adress of the events is calculated by simple integer division. So all the events in a subsampled patch are combined into one pixel which leads to temporal clusters in the generated spiketrains meaning that all the subsampled events of a patch will be fed into the same neuron of the network at

the same simulation-timestep. This strongly contrasts with the spiketrains obtained using poisson generators or analog input, where the resulting spikes are equally distributed.
Keeping only one of the subsampled events in a patch removes the temporal clusters in the input which increased SNN accuracy by 4.1%.

Non-uniform DVS spiketrains

As can be observed in the first row of Fig. 9, the distribution of the DVS spiketrains is beyond from being uniform, which contrasts strongly with the spike distributions observed using poisson or analog inputs. Strong temporal patterns can be observed in all layers when feeding in DVS data. However, as the ANN model was trained with static frames only, the model was never confronted by the inherent timing statistics of the DVS samples during training. So the SNN never had the opportunity to learn how to process the information encoded in the timing of the DVS spiketrains.

While this intuitively explains the remaining loss in accuracy, this can be proven by changing the input to poisson spiketrains while restricting the number of input spikes per sample to 5000 to match the spike count of the DVS samples. This restriction is important as using higher spike counts would lead to an unfair advantage when using poisson input. By doing so accuracy improved from 71% to 78.25% almost closing the gap to ANN accuracy. The remaining accuracy loss was now mainly due to the restriction induced on the number of input spikes per samples. By removing this restriction when generating the poisson spiketrains, accuracy improved to 80.15%, so the accuracy loss of the SNN was now at only 1%.

This result proves the initial presumption that the non-uniform temporal distribution of the DVS spiketrains and the low spiking-rates are responsible for the remaining performance gap between ANN and SNN.

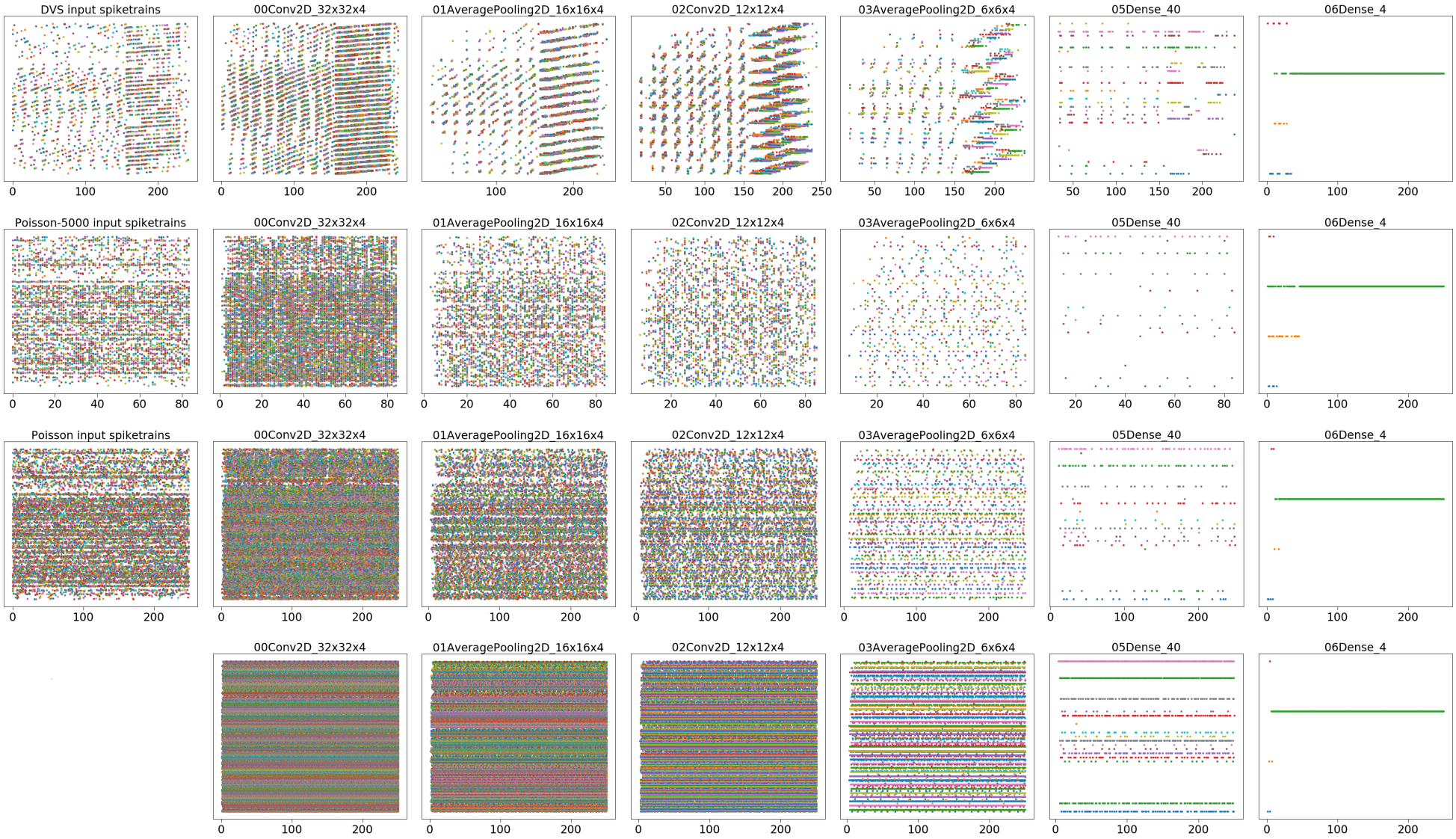


Figure 9: Spiketrains of all layers of the spiking CNN for DVS, Poisson and Analog input. The different inputs are ordered row-wise: 1. DVS, 2. Poisson with 5000-event restriction, 3. Poisson with no restriction, 4. Analog. The maximum-rate for the poisson spikes was set to 200Hz. X-axis: Simulation timestep, Y-axis: Neuron index

3.4.3 MLP conversion

The MLP network described in section 2.4 with one hidden layer of 16 neurons was also converted into its SNN counterpart, using the same setup as for converting the CNN architecture. Just as in the CNN conversion a loss in accuracy was observed when using DVS input.

Table 6 lists the measured accuracies for the converted MLP spiking network. A loss of only 4% was achieved using DVS input. The interpretation of the results is equivalent to the previous section.

	Accuracy	**Accuracy loss**
ANN	75.5%	-
SNN (Analog), p=99.9	76.75%	+1.25%
SNN (DVS)	70.0%	-5.5%
SNN (DVS), single subsampling	71.5%	-4.0%

Table 6: Accuracies for the converted MLP model using analog input using weight normalization with the 99.9 percentile activation and DVS input with standard subsampling and single event subsampling, where only one event of a subsampled patch is kept.

4 Implementing spiking neural networks on spiNNaker

4.1 Motivation

The results in the previous sections showed that ANN-SNN conversion works reasonably well with only small losses in accuracy even when using the sparse DVS spiketrains as input. However, using this approach to train SNNs there will always remain a certain loss in acuracy as the distribution of the data in time is completely neglected when training the ANN with artificially generated event frames. By training a SNN directly using event-based data with back-propagation like algorithms potentially enables the network to learn the particular timing patterns in the data. In recent work such algorithms have been proposed achieving state-of-the art accuracies in standard classification tasks [12], [19], [20]. However, there is a lack of neuromorphic hardware supporting efficient computation of these algorithms, and training in simulation on traditional computer architectures is computationally not feasible when it comes to training complex deep-learning architectures.

In this section the converted MLP spiking architecture from section 3.4.3 was implemented on the neuromorphic hardware platform spiNNaker [7]. The goal was to evaluate the suitability of the hardware for training SNNs directly using event-based data as well as analyzing computational performance during inference. As a starting point the trained MLP architecture and not the CNN was used due to its simpler implementation.

4.2 The spiNNaker platform

SpiNNaker [7] (Spiking Neural Network architecture) is a massively parallel computer system designed to provide a cost-effective and flexible simulator for neuroscience experiments. The basic building block is the SpiNNaker chip which consists of an array of ARM9 cores, communicating via packets carried by a custom ansynchronous interconnection fabric [22].

The spiNN-3 platform [27] used in this work comes with four spiNNaker chips where each of the 18 cores inside a chip has associated comes with 64Kbytes of data tightly-coupled memory (DTCM) and 32Kbytes of instruction tightly-coupled memory (ITCM) [7]. The board also contains an off-die 128 Mbyte (1-Gbit) SDRAM module. The master chip has an Ethernet interface provided via an external PHY chip which can run at 10 or 100 Mbit/s.

One spiNNaker chip can simulate up to 18.000 neurons (1000/core) and the routing fabric can handle about 1000 synapses per neuron. Each processor core is programmable and can implement any model that fits its 32 kB instruction memory.

The communication within a spiNNaker chip works purely digital, each spike is encoded by a 40-bit packet containing a 32-bit identifier of the source neuron. Each core within a chip is connected asynchronously with six neighbouring cores. However, the topology of the simulated neural network, is completely independent of the physical hardware design.

4.3 Implementation of a MLP on spiNNaker

SpiNNaker provides a high-level pyNN front-end interface called spinnaker8, which allows to map neural networks implemented in pyNN (0.8) to the spiNNaker board. The MLP architecture from 3.4.3 was implemented with this interface using the weight parameters produced by the ANN-SNN conversion toolbox. So the work-flow was as follows:

1. Training of the MLP (ANN) using DVS frames

2. ANN-SNN conversion using the toolbox

3. Implementing the SNN architecture in pyNN using the weight parameters generated by the ANN-SNN conversion toolbox

4. Mapping the pyNN implementation to spiNNaker using the spinnaker8 interface

In conrast to the SNN simulations performed in the last sections using the ANN-SNN toolbox, where a simple integrate and fire (IF) neuron model was used, the network on spiNNaker was implemented using leaky IF neurons. This way it was possible to feed in multiple consecutive samples at once while letting the membrane potentials decay to zero before applying the next sample (Fig. 10). This has found to be more efficient than applying the samples separately and manually resetting the membrane potentials after applying each sample. Furthermore a simple ReLU layer was used as output stage, whereas the toolbox uses a spiking softmax representation. To perform predictions the spikes produced by the four output neurons are counted and finally the class corresponding to the neuron that fired the most is predicted.

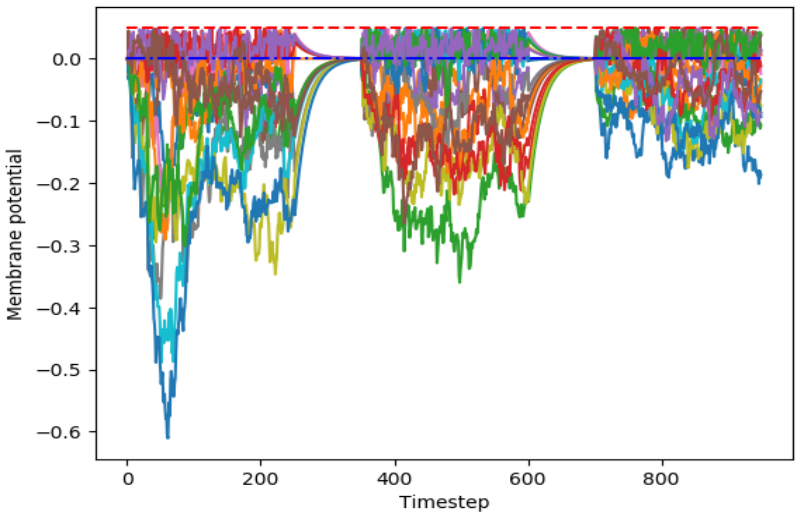


Figure 10: Membrane potentials of the 16 neurons in the hidden layer while applying three consecutive samples, letting the potentials decay to zero before applying the next sample.

4.4 Results

Two MLP models were trained and implemented on spiNNaker where the first model was restricted to positive weights and zero bias while in the second model MLP the positive weight constraint was released. The positive weight constraint was first introduced because of the simpler implementation and also because a simple ReLU output stage was used for the predictions. Using ReLU as output can lead to problems when membrane potentials get too negative and the output neurons thus don't produce enough spikes anymore to perform reliable predictions. However, when using strictly positive weights the resulting membrane potentials will also be always positive, so in this case omitting the softmax implementation is not an issue.

The achieved accuracies are listed in Table 7 where they are also compared to the accuracies achieved using the simulator of the ANN-SNN toolbox. While the model with positive weights achieved almost the same accuracy, a small loss in accuracy occurred using the second model. This loss has found to be mainly due to the missing spiking softmax implementation.

	spiNNaker	**ANN-SNN toolbox**
MLP, W>0, B=0	67.5%	68.25%
MLP, B=0	61.75%	70.0%

Table 7: Acchieved accuracies on the 400-sample testset for the two trained MLP models on spiNNaker and using the simulator from the ANN-SNN toolbox.

4.4.1 Performance

Performance in terms of inference runtime was slower than expected. This was mainly due to the limited memory resources of the spiNN-3 board. Loading the whole testset at once onto the board led to memory overflows. So the testset was loaded onto the board in batches. A batch is represented by one long spiketrain containing a sequence of all batch samples with enough space in between to let the membrane potentials decay to zero before a new sample is fed in. A batch-size of 30 has found to perform best.

However, each time a new batch of samples is loaded, a time consuming mapping operation takes place leading to huge overheads which compromises the performance. This might be circumvented by using a platform with more fast on-board memory resources and by using a more low-level implementation of the MLP instead of performing the mapping with the pyNN front-end.

For efficiency reasons the gaps in the DVS recordings were removed such that in each simulation step new input spikes are injected into the network (Fig. 11). By feeding in DVS events with similar timestamps at the same simulation step, performance was further improved. Combining the events with ten consecutive timestamps lead to good results without decrease in accuracy.
Still, using these methods the evaluation of the testset of 400 samples took about 5 minutes on average which is in the same order as the measured runtimes using the ANN-SNN simulator running on a i7-4770 CPU.

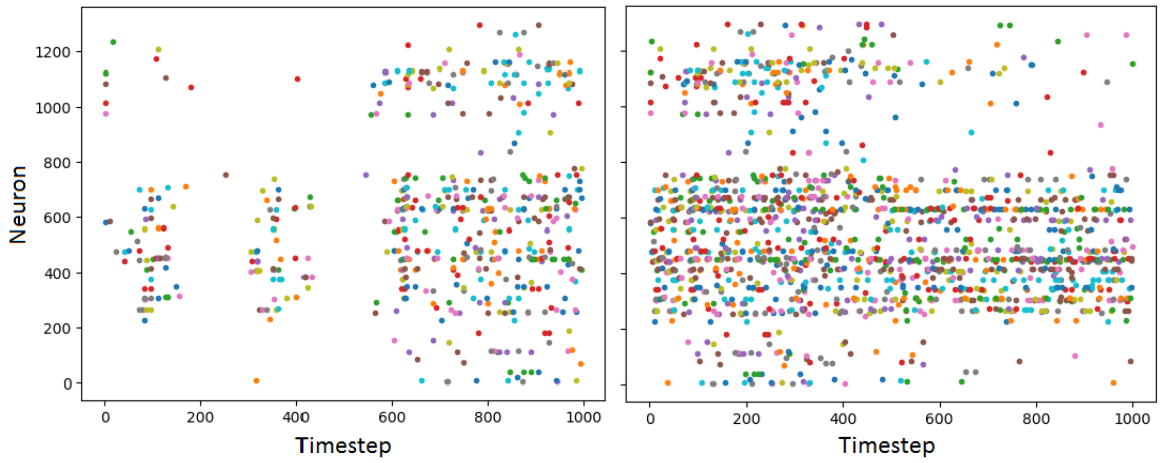


Figure 11: Original DVS spiketrains (left) and spiketrains with removed gaps for 1000 simulation steps. It can be observed that in the latter, much more input can be fed into the SNN during the same number of simulation timesteps.

4.4.2 Suitability for SNN backpropagation

When evaluating if the platform would be suitable for direct SNN training using backpropagation like algorithms, it was found that this is not possible using the high-level pyNN interface to map SNNs onto spiNNaker as the spinnaker8 interface currently does not support weight changes at runtime. However, a more low-level implementation would be feasible, but then still the question of how to resolve the memory bottleneck issue remains.

5 Conclusion and outlook

In this work the methods for ANN-SNN conversion as proposed by [25] could be successfully applied to the presented CNN and MLP architectures.

While very little loss (∼1%) in accuracy occurred when driving the converted SNN with synthetic input spiketrains and using normalized weights and biases to circumvent saturating firing-rates, for DVS input a bigger gap in accuracy with respect to to the ANN was measured. The conversion of the biases as well as the sparse and non-uniform nature of the DVS event streams were identified to be the main reasons for the observed loss after conversion.

The best result was achieved by converting a CNN with zero-bias constraint and modifying the subsampling operation for the DVS spiketrains to keep only one of the events in a subsampled patch. This led to a loss in accuracy of only -6.15% using the original event-based DVS input.

The resulting SNN architectures are fully event-based systems and this work showed that they can successfully be driven by purely event-based data such as acquired by neuromorphic vision sensors. In future work, the results could potentially be further improved by directly training the SNNs using backpropagation inspired algorithms instead of performing ANN-SNN conversion. Using this approach, the inherent timing statistics of the DVS data could be better captured. However, for efficient computation of these algorithms, dedicated neuromorphic hardware that supports the implementation of the required operations will be needed.

References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system.

[2] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[3] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95(1):1–19, 2006.

[4] Eugenio Culurciello, Ralph Etienne-Cummings, and Kwabena Boahen. Arbitrated address-event representation digital image sensor. *Electronics Letters*, 37(24):1443–1445, 2001.

[5] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[6] 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 *Rebooting Computing (ICRC), IEEE International Conference on*, pages 1–8. IEEE, 2016.

[7] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2013.

[8] INIlabs. Aedat fileformat. `https://inilabs.com/support/software/fileformat/`.

[9] Teijiro Isokawa, Haruhiko Nishimura, and Nobuyuki Matsui. Quaternionic multilayer perceptron with local analyticity. *Information*, 3(4):756–770, 2012.

[10] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. Hots: a hierarchy of event-based time-surfaces

for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1346–1359, 2017.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[13] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128× 128 120 db 15μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[14] Iulia-Alexandra Lungu, Federico Corradi, and Tobias Delbruck. Live demonstration: Convolutional neural network driven by dynamic vision sensor playing roshambo. In *2017 IEEE Symposium on Circuits and Systems (ISCAS 2017), Baltimore, MD, USA*, 2017.

[15] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4):593–616, 2004.

[16] Henry Martin and Jörg Conradt. Spiking neural networks for vision tasks. 2015.

[17] 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.

[18] Diederik Paul Moeys, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Delbrück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*, pages 1–8. IEEE, 2016.

[19] Emre Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Neuromorphic deep learning machines. *arXiv preprint arXiv:1612.05596*, 2016.

[20] Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324, 2017.

[21] Bernhard Nessler, Michael Pfeiffer, and Wolfgang Maass. Stdp enables spiking neurons to detect hidden causes of their inputs. In *Advances in neural information processing systems*, pages 1357–1365, 2009.

[22] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[23] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[24] Bodo Rueckauer. Spiking neural network conversion toolbox. `http://snntoolbox.readthedocs.io`.

[25] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv preprint arXiv:1612.04052*, 2016.

[26] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in neuroscience*, 11, 2017.

[27] Steve Temple. Appnote 1-spinn-3 development board. *SpiNNaker Group, School of Computer Science, University of Manchester*, 2011.

[28] Davide Zambrano and Sander M Bohte. Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv preprint arXiv:1609.02053*, 2016.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Event-based object recognition using analog and spiking neural networks

Verfasst von *(in Druckschrift):*
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Känzig	Nicolas Luca Ermano Max

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 18.12.2017	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.