Christoph Stöckl, Bsc

Image Classification with Spiking Convolutional Neural Networks

Master's Thesis
to achieve the university degree of
Diplom-Ingenieur
Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor
Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

Institute of Theoretical Computer Science
Head: Assoc. Prof. Dipl.-Ing. Dr. techn. Robert Legenstein

Graz, August 2019

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Abstract

The importance of artificial intelligence has significantly increased over the last years. There has been a lot of success in many different areas of machine learning, especially in computer vision, where AI has already surpassed human performance in some fields. The significance of computer vision can be seen clearly by its vast amount of applications, which include self-driving cars, face recognition, gesture recognition, robotics, image search, augmented reality and many more.
Nowadays most machine learning models used to solve these tasks run on graphics processing units (GPUs). Unfortunately, however, GPUs require a substantial amount of energy to operate, which can make them disadvantageous in numerous scenarios. As a result, there are many battery-powered devices, which are unable to make use of the benefits of AI as their energy budget does not allow for this kind of hardware.
One promising solution to this problem could be using spiking neural networks on neuromorphic hardware. Due to the event-driven nature of spiking neural networks, they are very economical with regards to energy consumption.
This master thesis will propose three spiking convolutional neural networks, which can be used for image classification and could also run on neuromorphic hardware.

Acknowledgements

I would dedicate this page to the people who stood by my side during my bachelor and my master study.
I am especially grateful for all the support I received from my parents, who enabled me to put a strong focus on my studies. Without their help, it would never have been possible for me to progress at this pace.

Of course, I also want to thank my brother, my friends and my girlfriend for their aid, encouragement and for proofreading my work. Being able to bounce ideas off them turned out to be very valuable for this master thesis.

Lastly, I also want to express my gratitude towards all the people working at the institute of theoretical computer science at the TU Graz for their support. I especially want to thank Prof. Wolfgang Maass, my professor and mentor, for his dedicated support and for introducing me into this fascinating field of research.

The LaTeXtemplate, developed by Karl Voit is available at: https://github.com/novoid/LaTeX-KOMA-template

Graz, September 2019

Christoph Stöckl

Contents

List of Figures

List of Tables

1 Introduction

Computer vision is a far-reaching category in artificial intelligence research. It is often the case that difficult tasks, such as autonomous driving or robotics, contain vision-related subtasks.
The area of computer vision contains many different vision-related tasks like image classification, image segmentation, object detection, image description and many more. Furthermore, computer vision is not only restricted to dealing with static images but it can also process video data.

This master thesis will mainly focus on image classification.
The goal of image classification is to teach a computer to assign a category to a digital image.
This is usually done by training a neural network in a supervised fashion. First, the network is shown numerous labeled images. The neural network tries to pick up on similarities, which images belonging to the same category have. Later, the model gets evaluated by showing it new images, which it has not seen before during the training phase. If the model managed to generalize well, it will be also able to classify unfamiliar images.
In recent years, a lot of progress has been made in this field. On some data sets, artificial neural networks have been reported to even surpass the performance of humans (Ho-Phuoc, 2018).

The idea of drawing inspiration from the human brain has always been a central concept of artificial intelligence research.
The human brain is a miraculous biological machine, which can solve very complex tasks effortlessly. It especially excels at problems involving one-shot learning or dealing with completely new tasks and situations.
Recent advances in hardware implementations of spiking neural networks, like Loihi (Davies et al., 2018), further boost research in this direction. Using special hardware, dedicated for spiking neural networks, has many advantages, like speed and power efficiency.

Being able to solve AI-related tasks, such as image classification, in low power environments could be seen as a milestone in porting AI to most devices running on batteries.

1.1 Overview

The first part of this master thesis will describe the state of the art of spike based image classification. The chapter 'Computer Vision' will give some insight into the task of image classification.
It will define the underlying problem and emphasize its importance. Furthermore, this chapter will introduce the two data sets used in this thesis and describe convolutional neural networks, which mark the state of the art for image classification tools.
The next chapter 'Spiking Neural Networks' will introduce the main ideas of spiking neural networks. It contains a brief overview of the motivations behind spiking neural networks and introduce mathematical models to simulate neurons.
In addition, there is also a section describing how spiking neural networks can be trained and a section underlining the importance of spiking hardware.
All chapters up to now focused on laying out necessary background information. The following chapter 'Spiking Convolutional Neural Networks' will be the main chapter of the thesis. Three different spiking convolutional neural network models will be introduced and described in detail. Plots depicting the spiking activity, as well as the performance of all models have been added.
The final chapter of this master thesis will briefly summarize the most important information. The final section contains an outlook into the future, laying out possible research directions.

2 Computer Vision

The goal of computer vision is to make computers 'understand' visual data. This is not only limited to digital images but can also refer to video data. There are numerous different tasks associated with computer vision. Some of them are:

- *Image classification*: The computer gets a collection of images and it has to assign a category to every image. There are usually somewhere between 2 to 1000 different categories.
- *Image segmentation*: The computer gets an image and it should decide which pixel belongs to which object. This is especially useful for self-driving cars, as it is important to know which part of an input image or video belongs to the road, traffic signs or obstacles.
- *Object recognition*: The computer gets an image containing potentially multiple objects. The task is to find and classify all the objects in a scene.

This master thesis will focus on the task of image classification.

2.1 Image classification

2.1.1 Importance

For some applications, image classification can represent the entire task. Examples for this category would include face recognition tasks, self-organizing galleries on smartphones or determining whether or not a patient has a certain disease based on an x-ray scan.
There are also many tasks where image recognition represents a subtask, like classifying road signs in self-driving cars.
Also in robotics image and video classification can be very important, for example to find the right objects to pick up or to navigate around obstacles.

Because of its importance, a lot of research has been directed towards image classification in the last decade.

2.1.2 Why neural networks?

The main goal of image classification is to teach a computer to assign a category to an image.
Although this task is trivial for humans, it is quite challenging for computers. What makes this task hard is the huge variety of images. For example, a car from the front looks very different compared to a car from the side or from the inside.
It turns out that it is very difficult to formulate a set of predefined rules, which can efficiently decide on the class of an image.
As classical programming can not solve this task, new ideas like neural networks are needed to tackle this problem.
Neural networks are loosely inspired by the structure of the brain. Especially the paradigm of many small computational units, which are connected in a network has successfully been adopted. However, artificial neural networks do not try to copy the behavior of the individual neurons.

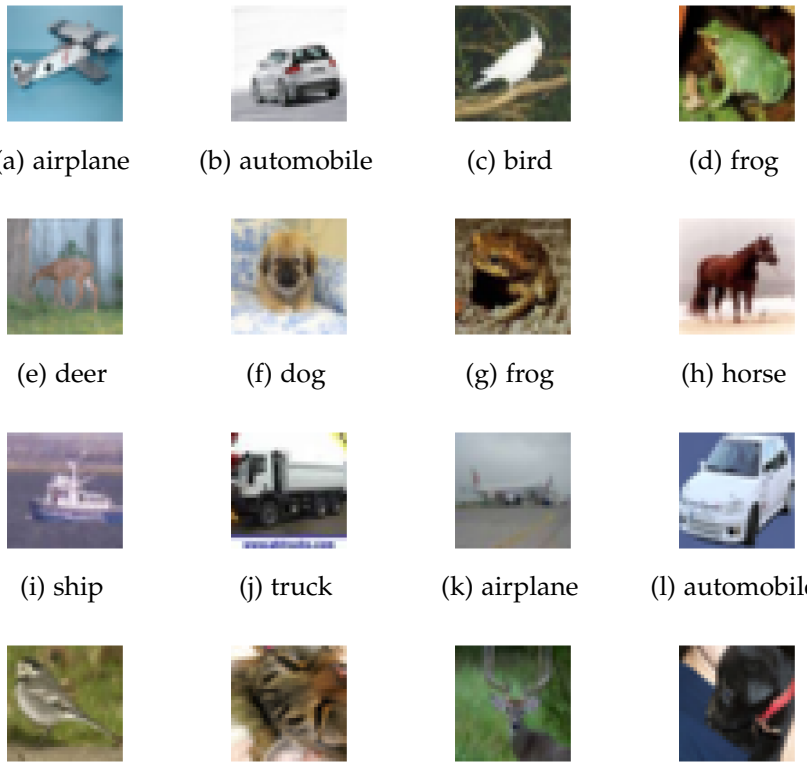
2.1.3 Data sets

Data sets play an important role in machine learning. For image classification, there exists a variety of different collections of hand-labeled images, which can be used to train, evaluate and compare models. These data sets differ in size and difficulty.
In order to reduce the computational complexity of the data sets, only low-resolution images were used as input for the models. However, using the models to classify larger images would certainly be possible.
Using low-resolution images is frequently done in image classification. Research has shown that using low-resolution images can be a fast way of finding the right hyperparameters, as training models can be done significantly faster. The hyperparameters obtained with low-resolution images tend to also work well for larger sized images (Chrabaszcz, Loshchilov, and Hutter, 2017).

The data sets used in this master thesis are CIFAR10 and a subset of the ImageNet data set.

CIFAR10

CIFAR10 is a data set consisting out of 60.000 low resolution, colored images (Krizhevsky, 2012). Every image has a resolution of 32 by 32 pixels. These images are split up into 50.000 training and 10.000 testing images.
The categories include animals (e.g. cat, dog, bird, frog, horse, deer) as well as modes of transportation (airplane, automobile, truck, ship). For every class, the data set contains 5000 training and 1000 test images.
Some sample images can be seen in figure 2.1.



(a) airplane (b) automobile (c) bird (d) frog

(e) deer (f) dog (g) frog (h) horse

(i) ship (j) truck (k) airplane (l) automobile

(m) bird (n) cat (o) deer (p) dog

Figure 2.1: Randomly selected sample images from CIFAR10

CIFAR10 can certainly be considered as one of the most widely used data sets for image classification.

ImageNet

ImageNet (Russakovsky et al., 2014) is one of the largest and most popular data sets used for computer vision. The original data set contains several million full resolution, colored and labeled images.
To reduce computational costs, it was essential to diminish the difficulty and size of the data set. This was archived by lowering the number of total classes to only 10 and reducing the image dimensions to 16 by 16 pixels.
The new classes were selected to be:

1. jaguar
2. ice bear
3. Indian elephant
4. tank
5. submarine
6. sports car
7. grand piano
8. beer glass
9. oscilloscope
10. basketball

These classes were chosen randomly.
The size of the data set is laid out in table 2.1.

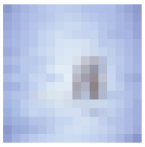
Number of training images	13000
Number of test images	500

Table 2.1: Size of the ImageNet16-10 data set.

As the training and test images are distributed evenly across the categories, every category has 1300 training and 50 test images.
Some sample images can be seen in figure 2.2.


(a) jaguar


(b) ice bear


(c) ice bear


(d) Indian elephant


(e) Indian elephant


(f) tank


(g) submarine


(h) submarine


(i) sports car


(j) sports car


(k) grand piano


(l) grand piano


(m) beer glass


(n) oscilloscope


(o) oscilloscope


(p) basketball

Figure 2.2: Randomly selected sample images from ImageNet16-10

As this data set is a randomly chosen subset of the ImageNet data set, there are no reported performances in the literature.
Therefore, on this data set human performance will be stated as a reference.

2.2 State of the Art

2.2.1 Convolutional Neural Networks

The introduction of deep convolutional neural networks marked a milestone in the area of image classification (LeCun et al., 1999). In the year 2012 it was shown that using a deep convolutional neural network architecture one could outperform all previous models (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012).
Over the next years, the architectures of the models became more and more refined and the state of the art was frequently redefined (Szegedy et al., 2015; He et al., 2015).
Convolutional neural networks consist of three basic types of layer, which will now be briefly described.

Convolutional Layers

Convolutions exploit a fundamental property of spatial information in images. The core idea is that every pixel carries more joint information with its neighboring pixels than with pixels that are spaced far apart. One reason supporting this claim is that neighboring pixels can encode visual features, like edges and corners.
Convolutional layers are designed in a way that they excel at detecting those visual features. Every convolutional layer contains a collection of feature maps (also called filters). Each feature map represents a special visual property, like an edge oriented in a certain way or a bright region surrounded by a dark region. These features are usually quite small, they are in the region of 3 by 3 to 7 by 7 pixels. The model uses the convolution operation to compare the visual feature map against numerous locations in the image, to see how strongly the feature is present in this area of the image.
This approach has many advantages compared to fully connected layers. Most importantly, convolutional neural networks can make use of weight sharing, enabling them to operate with a lower amount of trainable parameters. This is beneficial, as it allows the model to be larger in size. It is important to keep in mind that the models need to be run on hardware with limited resources, therefore using weight sharing can be very helpful. Fully

connected layers would not be a suitable solution for large scale images, as the number of weights needed would explode.
Figure 2.3 depicts the core concept of the convolution operation. On the bottom of the figure, there is a layer of neurons, on which the convolutional operation is applied. The blue cubes represent the neurons in the layer and the yellow cubes resemble neurons that are active.
The red and green rectangle represents the filter, which the convolution uses. This filter tries to pick up on straight lines. It achieves this by calculating a weighted sum over the neuron's activations and the filter's weights.
This means the output of the convolution using this filter will be the highest when the layer beneath shows an activity pattern that looks close to the green-colored part of the filter. A high output of the convolution results in a high activation to the corresponding neuron in the next layer, which will cause this neuron to also be highly activated.

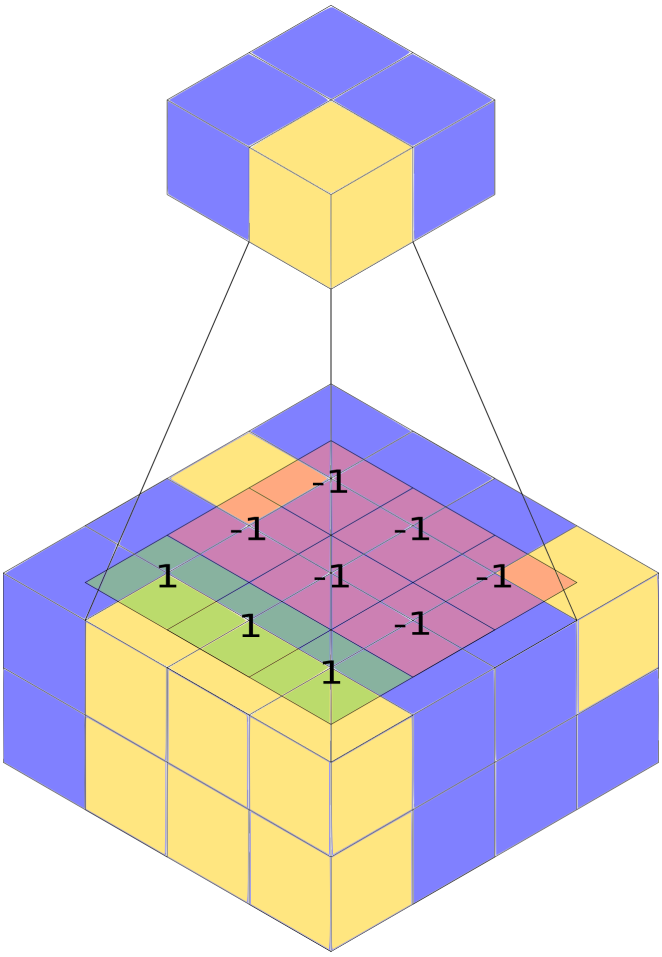


Figure 2.3: Visual concept of convolutions

Pooling Layers

As mentioned in the previous section, the filter sizes used in the convolution operation are rather small. Therefore the convolution will only be able to pick up on small visual features, as the features have the same size as the filters used to detect them.
In order to pick up on larger sized features, it is necessary to introduce a new layer type, which will compact the features found by a previous convolutional layer. These layers are called pooling layers.
As pooling layers reduce the size of the input, they can be combined with

convolutional layers to construct a hierarchical visual feature detection system.
The core idea of convolutional neural networks is that in the earlier layers the network picks up upon small features and then starts combining them into larger, more complex and more high-level features.
There exist multiple types of pooling:

- *Max pooling*: The highest activation present in a neuron in the pooling region will be propagated to the next layer. As the highest activation will be passed on, a minimum of information is lost in the pooling process.
- *Average pooling:* Average pooling calculates the average over a pooling region. The average value represents this region and it is passed on to the next layer. An average pooling layer could also be implemented using convolutions.

It should be mentioned, that CNNs can also work without pooling layers at all. The job of reducing the size can also be done by adding larger strides to the convolution. This means, that the convolutional filter is not applied at every position in the image but it skips some positions, resulting in a smaller size of the next layer.

Fully Connected Layers

In a fully connected layer, every neuron is connected to every other neuron in the next layer.
In convolutional neural networks, the fully connected layers can be found towards the end of the network. They receive a collection of high-level features present in the image and they combine this knowledge to make the final decision about the class of the image.

Output Layer

The last layer of a neural network is usually called the output layer. For every category in the data set there is a corresponding neuron in the output layer.

The activation of these neurons represents how much the model thinks that a given input image falls into the category represented by the output neuron.
Ideally, the final result of the neural network should be a discrete probability distribution, containing the probability of class affiliation for every category. In order to convert the real-valued activations of the output neurons into a probability distribution, the softmax function (equation 2.1) can be used.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.1}$$

3 Spiking Neural Networks

3.1 Why spiking?

Spiking neural networks draw their inspiration from biological neural networks. Among biological brains, the human brain is certainly among the most interesting subjects to study. It is capable of solving a rich variety of intelligence-related tasks, at which most machine learning approaches still struggle.
Some examples are:

- dealing with imperfect information
- one shot learning
- effortless image, audio and video classification
- structuring information
- abstraction
- memory

Many of those tasks pose enormous challenges for artificial intelligence.
Therefor, many researchers hope that studying the brain might help unlock ideas, which could be useful for understanding how the brain computes and also for building better AI models.
One approaches to tap into the potential of biological networks is by developing mathematical models of these networks and simulating them on computers. This way a lot of insight about the dynamics of biological neural networks can be gained.

3.2 LIF Neurons

The fundamental building block of biological neural networks is the spiking neuron. To simulate biological networks, it is important to find a suitable mathematical model to describe the behavior of the spiking neurons.

There exists a variety of different models. Unfortunately, there seems to be a trade-off between the biological realism and the computational complexity of the models.
The Hodgkin–Huxley model (Hodgkin and Huxley, 1952) for example is a very biologically accurate model but it comes with the price of having to solve four ordinary differential equations.
A very frequently used model, which seems to present a good compromise between biological accuracy and computational complexity, seems to be the leaky integrate-and-fire (LIF) neuron model.
This model assigns a so-called *membrane potential* u to every neuron. The membrane potential can be increased if the neuron receives positive external current inputs and it tends to decay back to a resting potential u_{rest} after some time. It can be described with the differential equation:

$$\tau_m \frac{\partial u}{\partial t} = -(u - u_{rest}) + R_m \cdot I \tag{3.1}$$

This differential equation holds until the membrane potential u surpasses a certain threshold value. When this happens the neuron *spikes*, which means that it emits a signal to all postsynaptic neurons, which are connected to it. After a spike the membrane potential u is reset to u_{rest}.
In biological neurons, there is a period of time right after a spike occurred, in which the neuron is unable to spike again. This period is called the refractory period and it has to be taken into account when trying to develop a precise model of a biological neuron.

3.3 Training

The most important part of working with neural networks of any kind is optimizing the network to solve a specific task. This process is called *training*.
There exists a variety of different training algorithms. Some of the algorithms consider aspects like spike timings, others contain an error module scheme.
A large portion of learning algorithms is gradient-based. These algorithms work by treating the network as a complicated mathematical function, which should be optimized. The core idea is to calculate the gradient of a function, which expresses the network's performance, with respect to the weights in

the model. This function is referred to as the *loss function* and it summarizes how well the network performs on the task with a single scalar number. Once this gradient has been obtained it is possible to change the weights slightly into the opposite direction of the gradient, as this will cause the loss function to decrease. This process can be repeated multiple times until the performance of the network stops increasing.
Obtaining the gradient can be done with *backpropagation*, one of the most widely used algorithms in machine learning (Rumelhart, Geoffrey E. Hinton, and Williams, 1986).
Unfortunately, backpropagation requires the network to consist of continuous computational elements, as it has to be possible to calculate the first derivative for every element. This is problematic when applying backpropagation to spiking neural networks, as the spike function is not continuous. As a consequence, the derivative of the spike function is a delta Dirac impulse, which unfortunately is not very useful as the only values returned by this function are zero or infinity.

To solve this challenge, an artificial gradient of the spike function can be introduced.
Figure 3.1 shows the function used to approximate the gradient of the spike function (Courbariaux et al., 2016, Essera et al., 2016).
The figure 3.1 clearly shows that the value of the pseudo gradient will be zero if the membrane potential moves too far away from the threshold. This should to be avoided and kept in mind when training the networks.

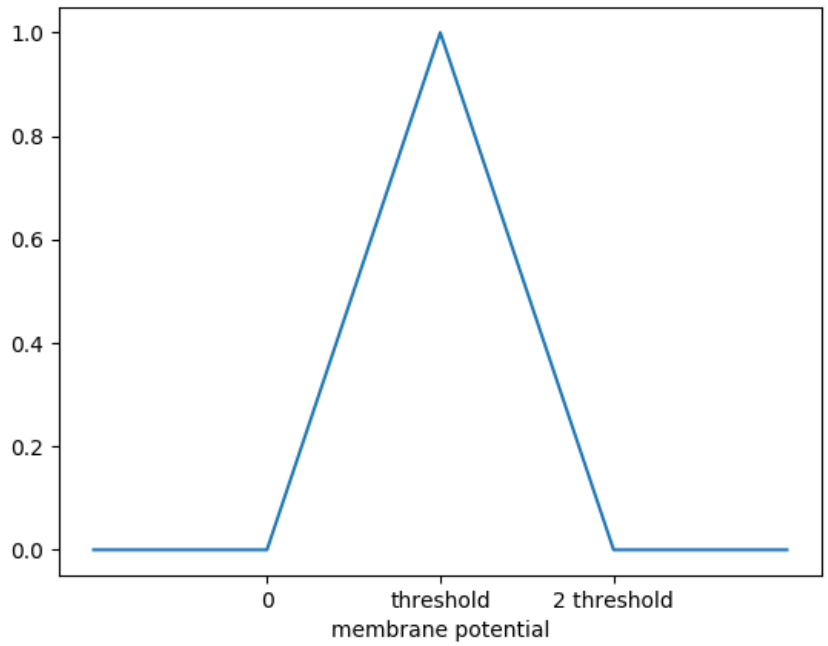


Figure 3.1: Artificial derivative of the spike function.

There are many proposals for different artificial gradient functions, such as box functions or functions shaped like Gaussians. However, it was found out that the shape of the artificial gradient does not seem to play a major role (Wu et al., 2018b).

3.4 Neuromorphic Hardware

In recent years there have been numerous approaches to implementing spiking neural networks in hardware.
One example would be IBM's TrueNorth (Cassidy et al., 2016), which has been reported to perform well on tasks such as object detection, classification and localization.
One of the most recent and notable approaches is Loihi, which is being developed by Intel.

A single Loihi chip contains a manycore mesh comprising 128 neuromorphic cores, which can manage up to 1024 neurons each. This way a single Loihi chip can handle roughly 130.000 neurons. It is also possible to increase the number of neurons in a neuromorphic system by combining multiple Loihi chips.
In one of the first papers introducing Loihi (Davies et al., 2018), Intel's neuromorphic research team shows, that Loihi is already capable of solving least absolute shrinkage and selection operator (LASSO) optimization problems with over three orders of magnitude improved energy-delay product.
These results show that spike-based networks on neuromorphic hardware can outperform all other state of the art solutions.
A very important feature, which a neuromorphic platform has to support in order to be able to run convolutional networks is weight sharing. Fortunately, Loihi also supports weight sharing, making it an attractive platform to run spiking convolutional neural networks on.

4 Related Work

This chapter is intended as an overview of the field of image classification with spiking convolutional neural networks.

4.1 State of the Art

The current state of the art test accuracies for both MNIST and CIFAR10 can be found in table 4.1

Data set	ANN	SNN
MNIST	99.79%	99.12%
CIFAR10	99% (Huang et al., 2018)	92.37%

Table 4.1: State of the art performance on the CIFAR10 and MNIST data sets

4.2 Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks

4.2.1 Training Algorithm

This paper (Wu et al., 2018b) introduces a training algorithm to train spiking neural networks in both the spatial domain as well as in the time domain. They argue, that in order to tap into the full potential of spiking neural networks one has to exploit the rich temporal features which SNNs provide. The neuron model used in this paper is the leaky integrate-and-fire (LIF) model, which is described in equation 4.1.

$$\tau \frac{\partial u(t)}{\partial t} = -u(t) + I(t) \tag{4.1}$$

They solved the differential equation to receive the following update rule:

$$u(t_i) = u(t_{i-1})e^{\frac{t_{i-1}-t_i}{\tau}} + \hat{I}(t_1) \tag{4.2}$$

Using the update rule defined in equation 4.2, it is easy to see that the membrane potential $u(t)$ only depends on the membrane potential at the previous timestep t_{i-1} and the pre-synaptic input current $\hat{I}(t_i)$.
They continue to derive the loss function with respect to the membrane potential for any given neuron.
This can be later used to obtain the loss with respect to the weights and the bias.
As the membrane potential depends on the previous membrane potential, it has to be taken into account when calculating the derivative of the loss with respect to the membrane potential. This is what they mean by propagating the error backward in the time domain.
As the membrane potential also depends on the pre-synaptic input currents it will also be part of the derivative. Considering the pre-synaptic input currents is what they mean by backpropagation through the spacial domain.
They introduce four types of approximated derivatives for the spike function and use backpropagation to calculate an error signal.
The approximated derivatives include a rectangle function, a triangle function and two differently shaped functions that are close to Gaussians. However, they report that the choice of the approximated derivative does not have a big impact on the resulting test accuracy.

4.2.2 Network

The Network used for the MNIST data set is a three layer fully connected network with layer sizes of 784, 400, 10. No convolutional layers are used in this model.
They also tested the CIFAR10 data set, but this time they used a different model with two convolutional and two fully connected layers.
The input convention used works on a probabilistic basis. Depending on the normalizes pixel value (values between 0 and 1) the pixel value is the probability that there will be a spike in the input layer at every time step. In other words, a pixel value of 0.8 will result in an 80% chance of a spike being triggered in the corresponding input neuron.

4.2.3 Results

The results can be found in Table 4.2. The paper mainly focuses on the results on the MNIST data set and only mentions the CIFAR10 results in the discussion section. They also do not report any training techniques or data argument methods.

Data set	SNN
MNIST	98.89%
CIFAR10	50.7%

Table 4.2: Performance on MNIST and CIFAR10

4.3 Paper: Direct Training for Spiking Neural Networks: Faster, Larger, Better

Continuing their work from the paper Spatio-Temporal Backpropagation the same research team published a follow-up paper (Wu et al., 2018a). They focused on scaling up their training algorithm to larger networks and achieving a better test accuracy on the CIFAR10 data set.
They also introduce a neuron normalization method which they call NeuNorm. The reason for this normalization is that they claim that too many spikes can harm the effective information representation.
However, their results show that using NeuNorm does not have a drastic impact on the performance (less than one percent).

4.3.1 Network

The network layout of the network used on the CIFAR10 data set can be found in table 4.3.
Usually with SCNNs of this size, the limited amount of memory on the GPU will become a big issue.
The reason why they can afford a network of this size is that they reduced simulation time to only 4 to 8 time steps. To cut down on the simulation time, they use a spike transmission without delay and they adapted the input and output convention.

Layer Number	Layer
1	Input
2	3x3 conv, 128 dropout
3	3x3 conv, 256, AP2, dropout
4	3x3 conv, 512, AP2, dropout
5	3x3 conv 1024, dropout
6	3x3 conv 512, dropout
7	1024 Fully connected
8	512 Fully connected
9	Voting layer
10	Output

Table 4.3: Architecture of the largest proposed model

4.3.2 Input and Output Convention

In a time window of size T a neuron can spike up to T times. As it is also possible for a neuron to not spike at all there are $T + 1$ different spike trains with distinct lengths. These spike trains can be used to encode $T + 1$ different input values. As it has to be possible for a neuron to spike in two consecutive time steps, the refractory period of the neuron is set to 0.
Using this input convention, an analog pixel value can be transformed into a spike train.
The output convention uses a voting strategy.
The last layer contains several neuron populations and each class is represented by one population. In this way, the burden of representation precision of each neuron in the temporal domain is transferred to the spatial domain. They claim, that this also reduces the simulation time.

4.3.3 Training

The loss function used is a least-squares error function. As the default loss function for classification tasks is usually cross-entropy, this is a usual choice.
The network was trained using Spacio-Temporal Backpropagation. NeuNorm was also used but it did not yield a significant performance improvement.

4.3.4 Results

The results can be found in table 4.4

Data set	SNN
CIFAR10	90.53%

Table 4.4: Performance on CIFAR10

The test accuracy reported on CIFAR10 is the state of the art for a directly trained spiking neural network on this data set.

4.3.5 Discussion

An impressive part of the paper is that they managed to train a very large SCNN of roughly 330000 neurons. The key to doing this was to reduce the number of required time steps to a very low number of around 4 to 8. Trying to train a network of this size with 50 to 100 timesteps will result in a memory consumption surpassing 100 GB.

4.4 Approaches using ANN to SNN conversion

Another paradigm for creating spiking neural networks is converting a trained ANN to an SNN. The main idea is to first train an ANN and then try to approximate this ANN with an SNN.
In the papers (Hu et al., 2018; Sengupta et al., 2018) this is done by approximating the ReLU activation function used in the ANN with a spike rate in the SNN. They report very little accuracy loss caused by the conversion. In order for this conversion to be accurate enough very long simulation times are required, which is computationally expensive. In the paper (Sengupta et al., 2018) up to 2500 time steps have been used.
The main reason why these approaches are popular is the fact that good performance can be achieved. The paper (Hu et al., 2018) reports a test accuracy of 92.37% on CIFAR10, which is the state of the art for spiking convolutional neural networks.

4.5 Approaches using STDP

There is a number of spiking CNNs trained by Spike-timing-dependent plasticity that currently exist. (Masquelier and Thorpe, 2005; Wysoski, Benuskova, and Kasabov, 2008; Beyeler, Dutt, and Krichmar, 2013; Kheradpisheh, Ganjtabesh, and Masquelier, 2016). However, these networks usually only consist of one trainable convolutional layer and are small in size. Usually these papers use MNIST only as a data set. In one example (Tavanaei and Maida, 2017) of such a model, a test accuracy of 97.5% was reported. Other STDP based SCNNs were able to achieve a test accuracy of 98.4%.

5 Spiking Convolutional Neural Networks

The main goal of this master thesis was the development of three spiking convolutional neural networks, which were trained on a subset of ImageNet and CIFAR10.
All three networks explore different approaches and techniques, which can be used to implement spiking convolutional neural networks (SCNN).
The three models presented in this thesis are:

1. *Temporal spiking convolutional neural network (TSCNN)*: The main emphasis of this model is to make use of the time dimension of the spiking convolutional neural network. The input convention utilizes the time domain and the convolutional filters have been enabled to use the time dimension for computations. To make this possible, the model uses 3D convolutions to be able to capture features within the time dimension. Using 3D convolutions introduces the concept of spike delays in the network. This empowers the network to take spikes into account, which appeared several time steps ago. This network has been trained on a subset of the ImageNet database. Its input images are scaled down to a resolution of just 16 by 16 pixels. It is a rather lightweight SCNN.
2. *Wide spiking convolutional neural network (WSCNN)*: The central characteristic of the WSCNN is that the convolutional layers contain a large number of filters, which results in a large network consisting of an extensive amount of neurons and synapses. The core motivation behind this model is performance. Therefore it is considerably larger in size than the TSCNN. The WSCNN has been trained on the CIFAR10 data set.
3. *Residual spiking convolutional neural network (RSCNN)*: The recent years have shown a trend in CNN models to consist of more and more layers.

One innovative idea which made this possible was the introduction of residual connections (He et al., 2015). These connections skip some layers completely, which makes it possible to train deeper networks.

Furthermore, the timescale used in the TSCNN is different from the timescale used in the WSCNN and the RSCNN.
In the TSCNN, one time step refers to a 1 ms time interval. It is important to keep in mind that biological neurons can not spike arbitrarily fast. Right after a spike, they need a short period of time to recover. During this period it is impossible for the neuron to spike again. In order to keep biological realism, it is important to introduce a refractory period for each neuron, during which it can not spike again.
Both WSCNN and RSCNN use a different timescale for one time step. Here one time step refers to a period of roughly 10 ms. This means that there is no need for a refractory period, as neurons can spike in two consecutive time steps without violating biological limitations.
The following sections will describe each model in more detail.

5.1 Temporal SCNN

5.1.1 Architecture

A detailed description of the model's architecture can be found in the table 5.1 and figure 5.1. All neurons in this network are leaky integrate-and-fire neurons, except for the ten neurons in the output layer. These neurons do not spike and they are also not leaky.
The model utilizes 3D convolutions, as the time dimension has been added to the filters. This means that the convolutional layers try to extract spatio-temporal features.

num.	type	filter size	num. of filters or neurons
1	input layer	-	-
2	convolutional	4x4x4	10
3	max pool	2x2	-
4	convolutional	4x4x4	15
5	max pool	2x2	-
6	convolutional	4x4x4	20
7	max pool	2x2	-
8	fully connected	-	100
9	fully connected	-	10

Table 5.1: Layers of the TSCNN model

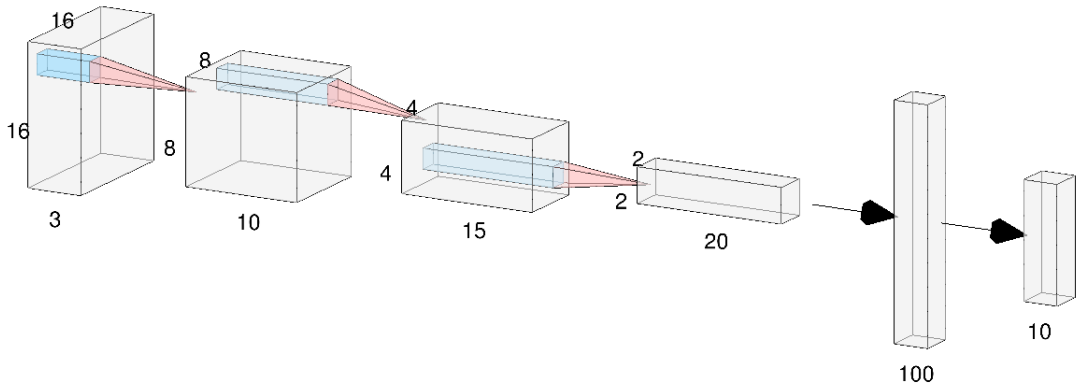


Figure 5.1: TSCNN network architecture

Additional information regarding the size of the model can be found in table 5.2

Number of neurons	1838
Number of weights	39720

Table 5.2: Number of neurons and weights in the TSCNN model

5.1.2 Input convention

For every pixel in the input image, there are three input neurons - one corresponding to every color of the pixel. The main idea of the input convention is to trigger a single spike after a delay which is proportional to the pixel value.
This way neurons with a low corresponding pixel value will spike later and pixels with higher values will cause their neurons to spike earlier.
It can be said that this input convention uses the temporal domain to encode the input image. For this reason, the convolutional filters have been extended to also include the time dimension.

5.1.3 Output convention

As the network is trained to solve an image classification task, the output of the network has to be a category.
There are ten different categories in the data set and each class is represented by one neuron in the output layer.
Initially, the number of spikes in the output neurons were used to determine the model's class choice. The neuron with most spikes would determine the prediction of the model.
However, it turned out that a small improvement in test accuracy can be obtained by changing the output convention from using spikes to using each output neuron's membrane potential directly.
With this insight, the output convention was changed so that the output neuron with the highest membrane potential at the last time step would decide on the model's prediction.

Lastly, a softmax function (2.1) has been applied to the membrane potentials of the output neurons. An advantage of this function is that it maps every component of an input vector into the interval from $(0, 1)$. As the output of the softmax function also adds up to 1 it can also be interpreted as a vector of class affiliation probabilities.
In figure 5.5 and 5.6 the change of the membrane potential of the output neurons over time can be seen.

5.1.4 Training

There are 1300 train images and 50 test images for every class. This makes a total of 13000 training and 500 test images in the data set.
The network is trained using an Adam optimizer (Kingma and Ba, 2014), which uses backpropagation through time for SNNs (Bellec et al., 2018) to obtain the gradients.
Cross-entropy was chosen as the loss function, which was minimized by the Adam optimizer. The membrane potentials of the output neurons were used as the logits, which were passed into a softmax function and then into a cross-entropy loss function.
Additionally, a regularization scheme was used, which will be described in detail in the following sections.
The weight initializations were sampled from a Gaussian probability distribution with a mean of 0.0 and a standard deviation of 0.001.

Parameters

The parameters used to obtain the reported results can be found in table 5.3. An extensive amount of hyperparameter tuning was used to obtain this set of parameters.

Parameter	Value
generations	50000
batch size	128
learning rate	0.00001
num refractory	5
num time steps	40
thr	0.01
membrane potential decay	0.8
thr coeff	5
initialization stddev	0.001
initialization mean	0.0
strech	1.1
input period	25
time conv depth	4
reg coeff	5

Table 5.3: Hyperparameters used for the TSCNN model

A detailed description of the parameters can be found in the appendix.

5.1.5 Spike Plots

Figure 5.2, 5.3 and 5.4 show the spiking activity throughout the network. As the model contains too many neurons it is not feasible to plot the whole spiking activity of the network. Therefore the spiking activity of only 50 randomly chosen neurons was illustrated in the following figures.

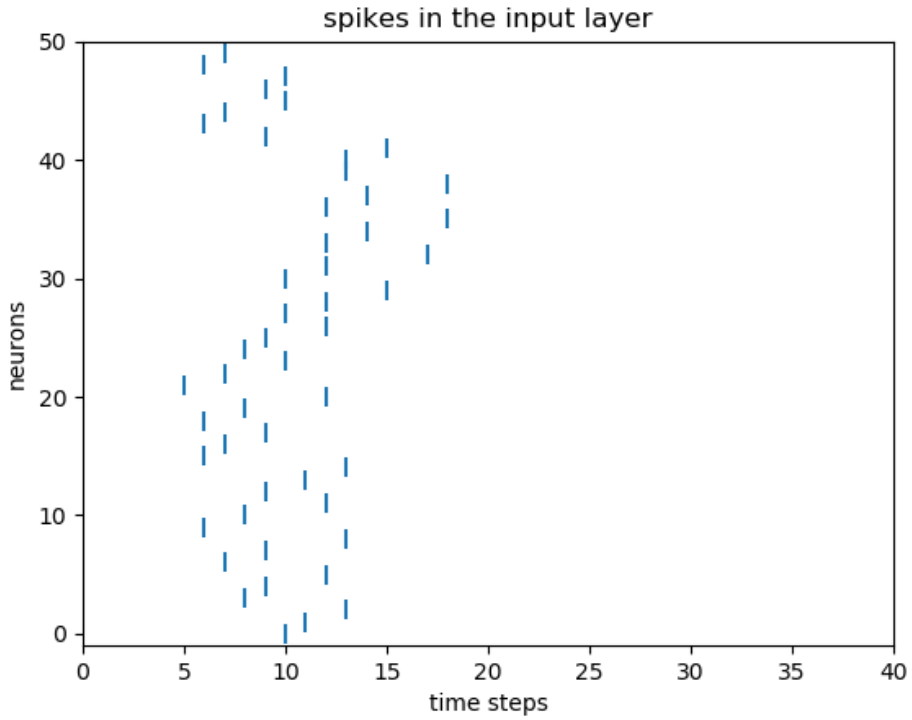


Figure 5.2: Spiking activity of the TSCNN model in the input layer

Figure 5.2 shows part of the spiking activity in the input layer. It can be observed that every neuron spikes once and only once due to the input convention used.

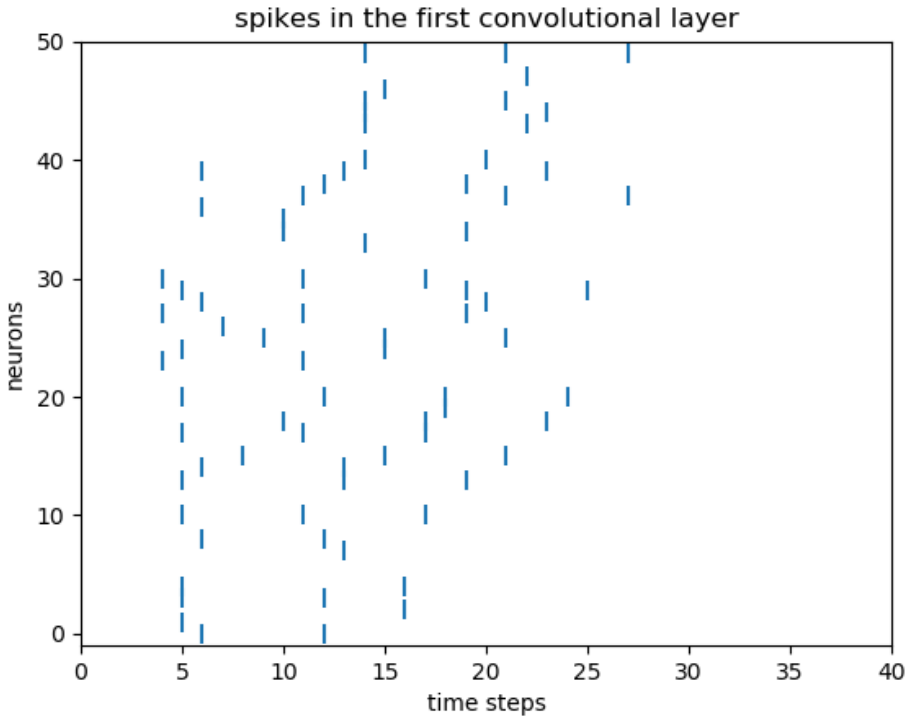


Figure 5.3: Spiking activity of the TSCNN model in the first convolutional layer

The spike plot in figure 5.3 depicts the spiking activity in the first convolutional layer. Here it is possible that neurons spike multiple times or not at all.

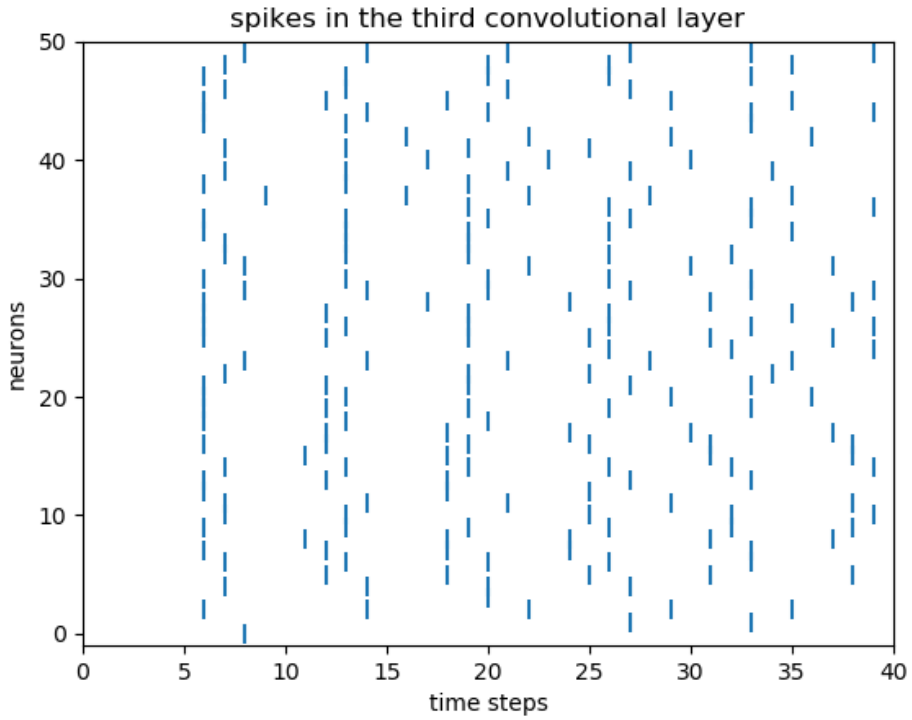


Figure 5.4: Spiking activity of the TSCNN model in the third convolutional layer

Figure 5.4 illustrates the activity of 50 randomly chosen neurons in the third convolutional layer.

5.1.6 Results

The classical version of the network, which uses artificial neurons instead of spiking neurons, is slightly better than the spiking version and it is very close to human performance. The TSCNN model was able to achieve a test accuracy of 57.59% on the ImageNet subset.
The value for the human performance has been obtained by having a single human label 200 different images. It should be added, that the human did not receive online feedback. Only after labeling all 200 images the total accuracy was revealed. At first glance, the human performance looks surprisingly low. It is important to keep in mind that the resolution of the

images is just 16 by 16 pixels, so even for a human, it can be tricky to decide which category an image belongs to.
The results also show that a decent test accuracy can be achieved with a very lightweight and computationally cheap model.

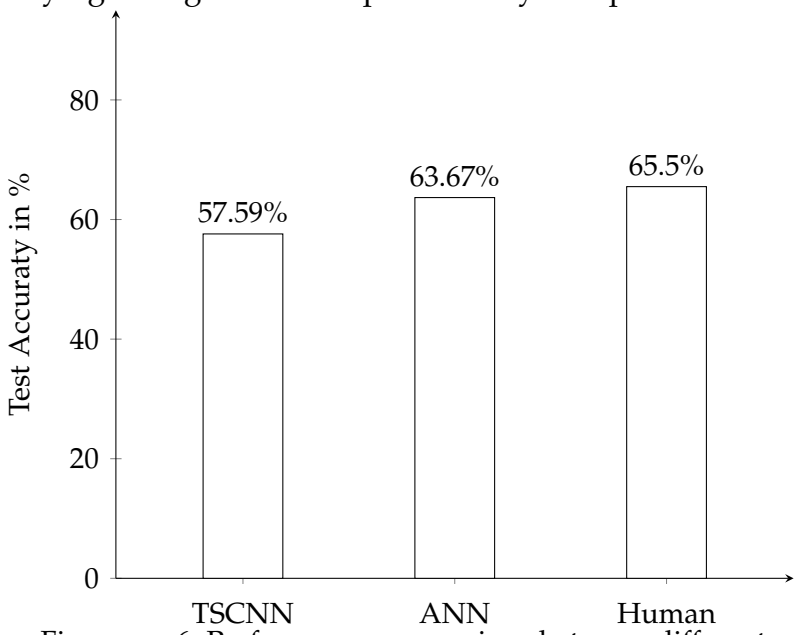


Figure 5.1.6: Performance comparison between different models

Figure 5.1.6 shows a comparison of the test accuracies of the TSCNN model, an ANN and a human. The spiking convolutional neural network manages to achieve 90.45% of the performance of the ANN and 87.92% of the human performance.
The performance of the artificial neural network (ANN) in figure 5.1.6 refers to a model which has the same architecture as the TSCNN, but consists of artificial neurons.
Figures 5.5 and 5.6 show the change of the membrane potentials of the ten output neurons over the simulation time. The model has been simulated over 50 time steps. In figure 5.5 it is clearly visible that the neuron representing class 6 ends up with the highest membrane potential, therefore the model would choose this category for this specific input image. This is the correct choice, as the right label is indeed 6.

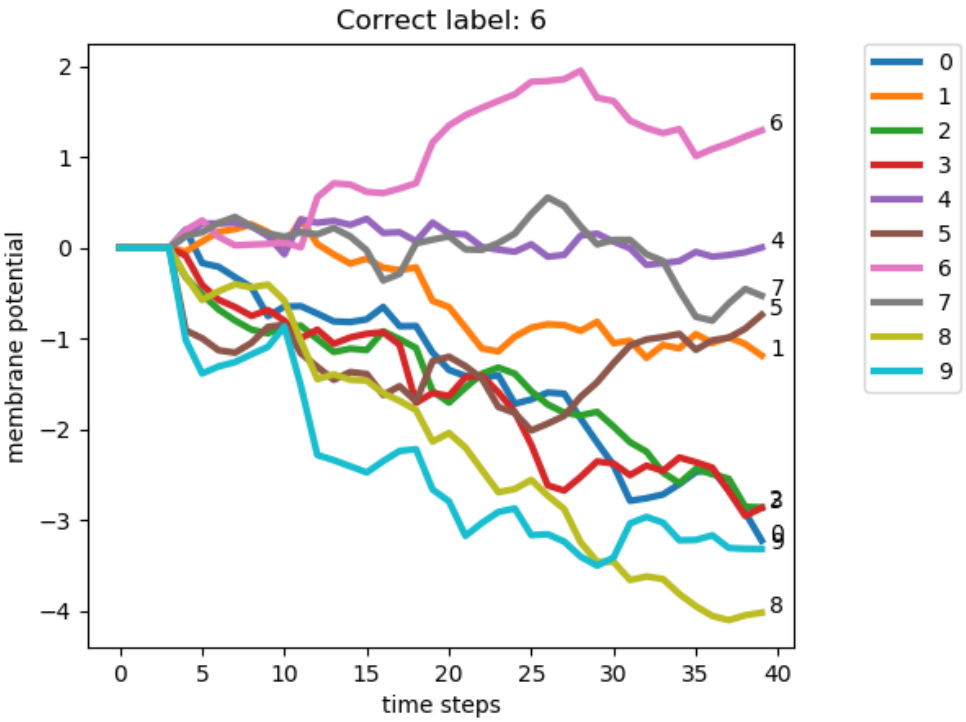


Figure 5.5: Change of the output neuron's membrane potential over time

The evolution of the membrane potential of the output neurons in figure 5.5 can be observed to be very volatile. The output neuron representing the class 6 has the highest membrane potential from the 15th time step onwards. After this point in time, the model becomes more and more certain in its decision.

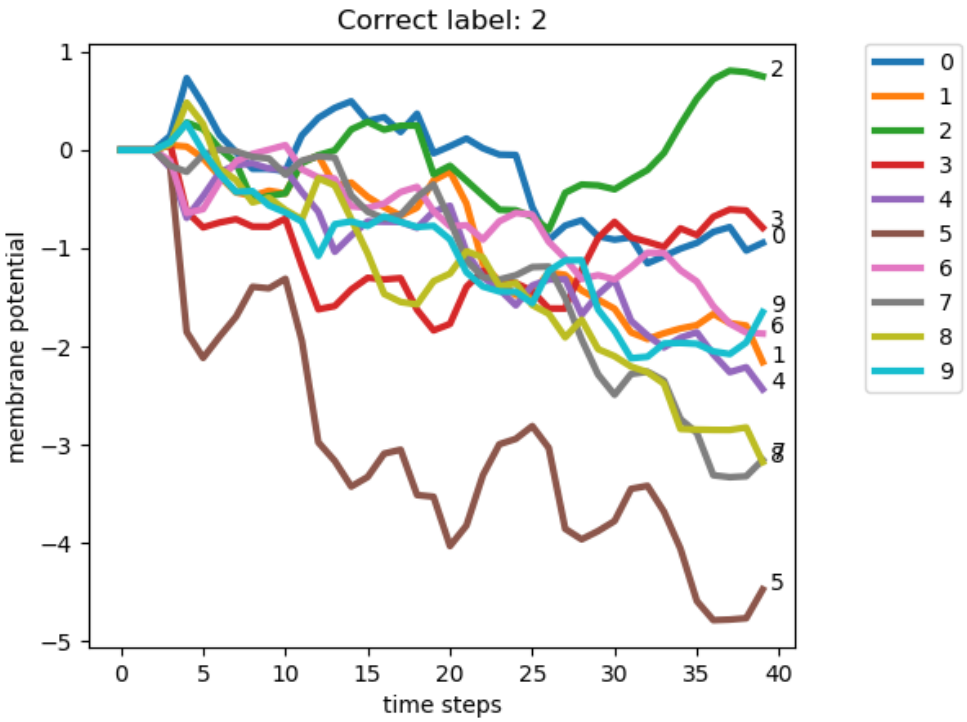


Figure 5.6: Change of the output neuron's membrane potential over time

It can be detected that in figure 5.6 the output neuron of the correct class only starts to stand out towards the end of the simulation. During the first half of the simulation time, the models' prediction would have changed quite frequently.

5.2 Wide SCNN

The main differences compared to TSCNN are the model's size, the data set used and a change in simulation timescales.
In contrast to the TSCNN model, more frequently utilized 2D convolutions have been used.
One time step accounts for 10 ms, which means that it is possible for a neuron to spike in two consecutive time steps.

5.2.1 Architecture

The model's architecture can be seen in table 5.4. The fourth column lists the number of filters used in the convolutional layers or the number of neurons used in the fully connected layers.
Figure 5.7 depicts the architecture of the network graphically.

num.	type	filter size	num. of filters or neurons
1	input layer	-	
2	convolutional	3x3	64 filters
3	pooling	2x2	
4	convolutional	3x3	128 filters
5	pooling	2x2	
6	convolutional	3x3	256 filters
7	convolutional	3x3	512 filters
8	convolutional	3x3	256 filters
9	fully connected	-	1024 neurons
10	fully connected	-	512 neurons
11	fully connected	-	10 neurons

Table 5.4: Architecture of the WSCNN model

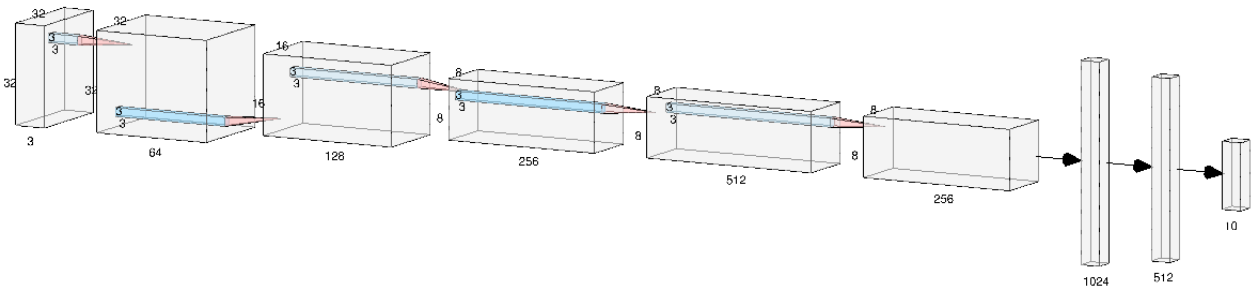


Figure 5.7: WSCNN network architecture

Compared to the TSCNN, this network is considerably larger, regarding both neurons and weights. It has roughly 87 times more neurons and 504 times more weights. The differences in size can be explained by both the larger architecture but also by the higher resolution of the input images. The exact size can be found in table 5.5.
The large size poses additional challenges regarding the computational complexity and the training of the spiking convolutional neural network.

Number of neurons	168.458
Number of weights	20.036.288

Table 5.5: Number of neurons and weights in the WSCNN model

5.2.2 Input convention

The WSCNN model was evaluated with two different input conventions. The first input convention uses probabilities to encode the input image. Every pixel is represented by three input neurons, where each neuron represents one of the pixel's primary colors.
First, the pixel values are normalized to have values between 0 and 1. Then, the pixel values can be interpreted as the probability that a given input neuron will spike on the current time step. High pixel values will, therefore, cause neurons to spike more often and low pixel values will tend to cause fewer spikes in the input neurons.
This input convention works well if there are enough time steps, as the average number of spikes will tend to converge to the pixel value as a consequence of the law of large numbers. However, when the simulation

only lasts a few time steps, it is good to adapt this input convention in a way that the probability gets replaced by some deterministic scheme.
The second input convention does this by turning the pixel value into a number of spikes. Every input neuron will spike every time step until the amount is reached. This will cause neurons representing high pixel values to spike more often, as well as longer and neurons with low pixel values to spike less often and stop sooner.
Using the second input convention leads to a 2% test accuracy improvement, so the rest of this thesis will be referring to this input convention.
This input convention is, therefore, both time and spiking frequency-based. Stronger pixel values will lead to later spikes in the corresponding neurons in the input layer. It will also lead to more spikes in a fixed time period, which increases the spiking frequency.

5.2.3 Output convention

All neurons in the network are LIF neurons - except the ten neurons in the output layer. They are special in the sense that they are not leaky and they also do not spike. The output neurons will only integrate over all incoming currents.
The membrane potentials at the last time step are interpreted as the output logits of the model. Therefore, the output neuron with the highest membrane potential will define the model's prediction.
As the last step, the membrane potentials of the output neurons are passed into a softmax function to obtain a class affiliation probability distribution.

5.2.4 Training and Hyperparameters

The model was trained using backpropagation through time in an end to end fashion.
Cross entropy was chosen to be the loss function and an Adam optimizer (Kingma and Ba, 2014) was used to perform the weight updates.
The learning rate used started at a value of 2e-5 and decayed linearly to 2e-7 during the whole training period.
The hyperparameters used to obtain the best results have been listed in table 5.6. An extensive amount of hyperparameter tuning was necessary to

obtain these values.
A detailed description of the parameters is provided in the appendix.

batch size	128
learning rate	2e-5 till 2e-7
Num. time steps	15
threshold	0.01
decay	0.45
dropout probability	25%
input noise stddev	0.01
initialization stddev	0.001
initialization mean	0.0
output decay	1
reg coeff	10
reg tolerance	0.012
stretch	1.05

Table 5.6: Hyperparameters used for the WSCNN model

Regularization

An important insight into training large spiking neural networks is, that it can be very beneficial to use a membrane potential regularization scheme. One of the problems that can occur when training spiking neural networks is that the membrane potentials tend to become more extreme during training. This has the consequence that the artificial derivative of the spike function will return zero for membrane potentials which are far away from the threshold. When trying to obtain the gradient of the weights with respect to the loss using backpropagation, this is problematic as receiving zero from the spike function's derivative will effectively stop the gradient at this point. This means that it is important to keep the membrane potential in a certain regime close to the threshold.
Regularization can achieve this by making this goal part of the objective during training. In other words, the loss function gets extended by a regularization term.
This term looks like a squared loss function, as can be seen in figure 5.8 (The shape of this regularization term was invented by Arjun Rao).

It can be seen that a membrane potential outside of the tolerance zone will result in a loss, so the optimizer will try to keep the membrane potential within the interval where the additional regularization loss is zero.
This way the membrane potentials are very likely to stay inside a regime where the value of the gradient of the spike function is non-zero.

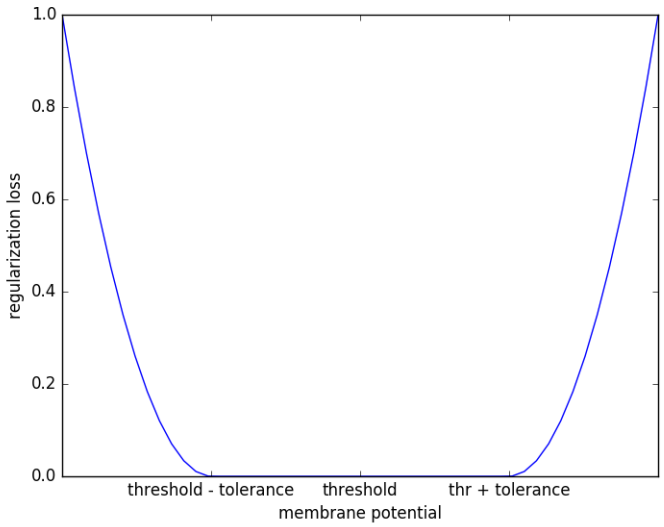


Figure 5.8: Regularization loss of a single neuron.

The extra regularization term is large when many neurons have a membrane potential which is far away from the neurons internal threshold value and it is small when the membrane potential is close to the threshold.
This way the network will try to decrease the classification loss, while also making sure that the membrane potential of an individual neuron does not become too extreme. Therefore gradients will be able to be propagated through the network even after many training iterations.
Without regularization, the gradients of the loss with respect to the spikes in one batch tend to have only 35% non-zero values after a couple of training iterations.
However, if regularization is being used this number stays roughly around 80% even after the training is finished.

5.2.5 Dropouts

During the training phase, the network becomes better and better at classifying the training images correctly. However, the main objective is not to have a good accuracy on the training data set, but the goal is that the network manages to generalize well and therefore is also able to show a high performance on images that it has never seen before. After numerous training iterations, it can often be observed that only the training accuracy increases but the test accuracy does not improve anymore. Sometimes the test accuracy can even decrease again. This undesirable phenomenon is called overfitting.
Using dropouts is a commonly used technique to reduce overfitting (Srivastava et al., 2014).
The core idea of using dropouts is to randomly remove some neurons during training by making them inactive. As a result, it is harder for the model to fit to the training data. This will make the model more robust, which has a positive impact on the test accuracy.
As spiking neural networks are simulated over a period of time, it is important to keep the same neurons dropped out at every time step (Lee, Sarwar, and Roy, 2019). Using different dropout masks for every time step will cause the dropout effect to average out over time, making it less effective. Commonly used dropout rates are often in the range from $20\% - 30\%$. However, also some more extreme dropouts up to 50% have been reported to work well (Wu et al., 2018a).

5.2.6 Image preprocessing

Two further techniques have been used to battle overfitting.
One of the most effective ways to combat the negative effects of overfitting is to increase the size of the data set. Unfortunately, this is usually not feasible. However, on image data sets, it is always possible to increase the size artificially by reflecting images over the vertical axis with a probability of 50%. Applying this operation will always result in a different valid image.
The second preprocessing step is adding a small amount of Gaussian noise to the image to ensure that the network will never receive the same input image multiple times. This usually also leads to a better generalization and therefore to an increase in the test performance.

5.2.7 Spike plots

Due to the size of the model, it is impractical to plot the spikes over all the neurons of the network.
To still get a feeling for how the spiking activity inside the trained model looks like 50 neurons have been picked randomly and plotted.

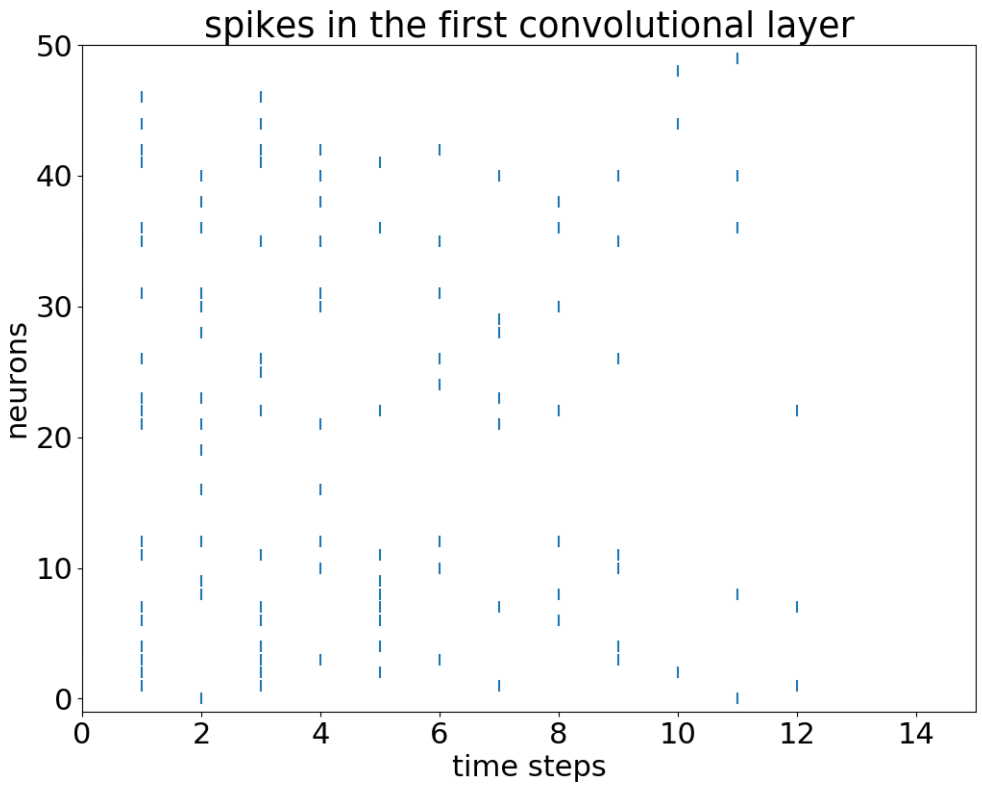


Figure 5.9: Spikes in the first convolutional layer

In the first convolutional layer, the spiking activity is very sparse. Some neurons have learned to spike with a regular period. This indicates that these neurons learned to operate with a rate-based information scheme. The spiking activity seems to decay towards the last time steps.

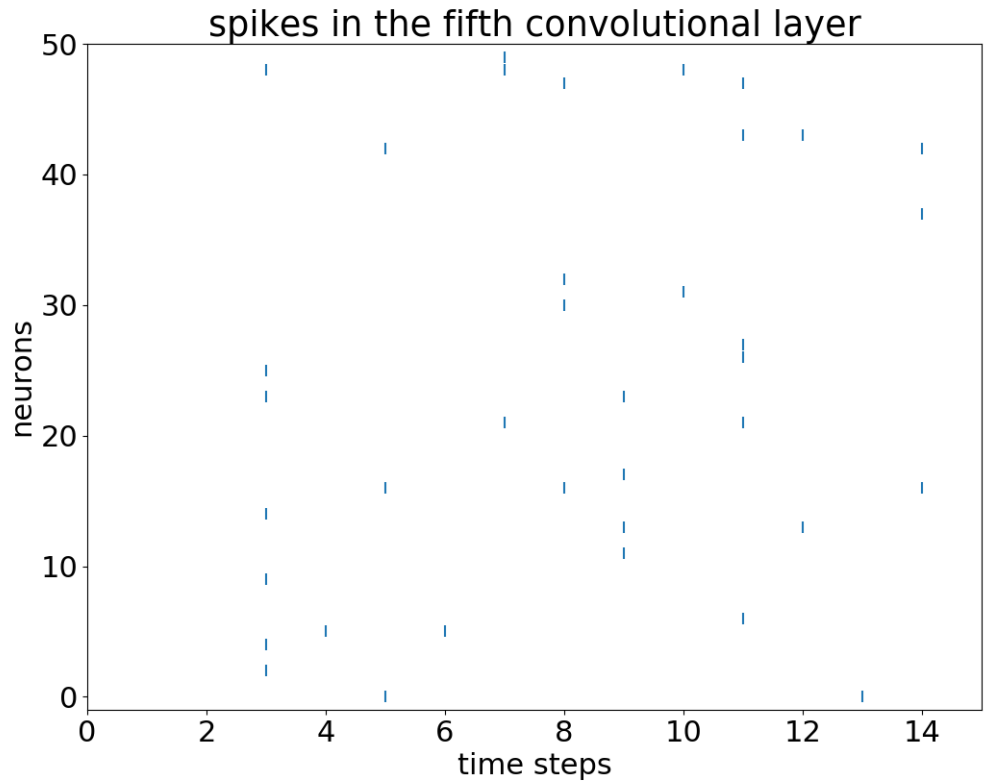


Figure 5.10: Spikes in the fifth convolutional layer

The fifth convolutional layer displays even less spiking activity than the first layer. A sparse spiking activity is especially desirable when running the model on neuromorphic hardware, as this will result in a lower energy consumption.

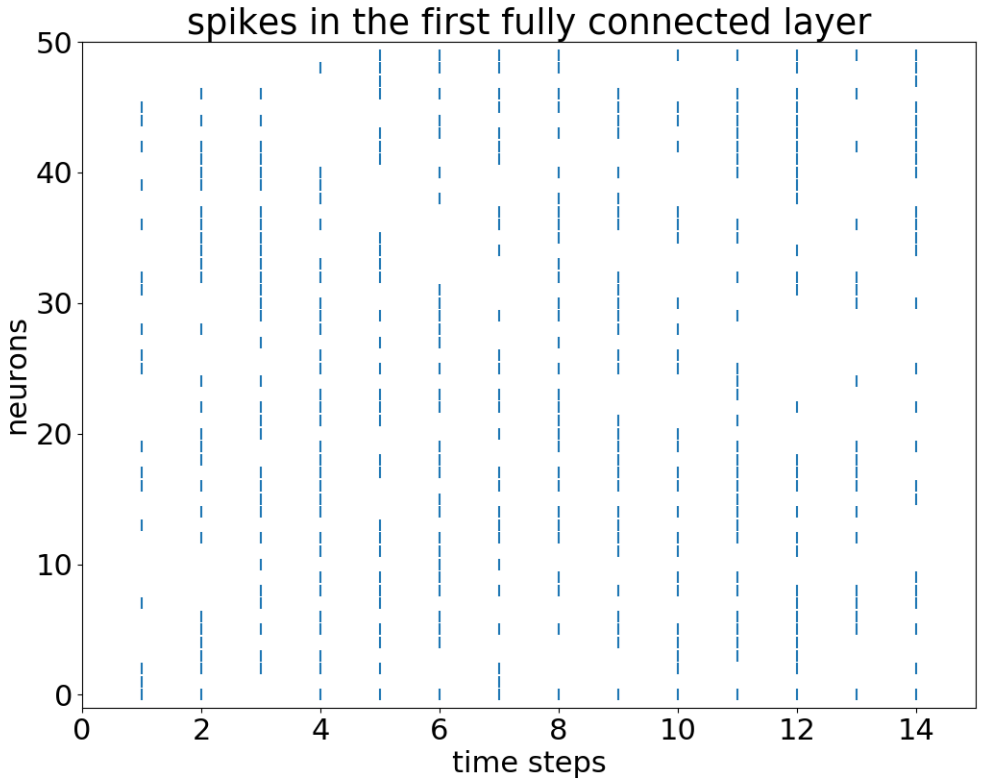


Figure 5.11: Spikes in the first fully connected layer

The spiking activity is very sparse in the convolutional layers but the neurons in the fully connected layers tend to spike more often.

However, since most neurons are part of the convolutional layers (more than 99%) the spiking activity of the whole model can be declared as very sparse.

5.2.8 Results

The test accuracy of the model is 85.19%.
In figure 5.12 and in figure 5.13 the membrane potential of the output neurons of two correctly classified examples can be seen. Note how the network becomes more and more certain with every consecutive time step.

It should be mentioned that these are two randomly selected samples which have not been cherry-picked.

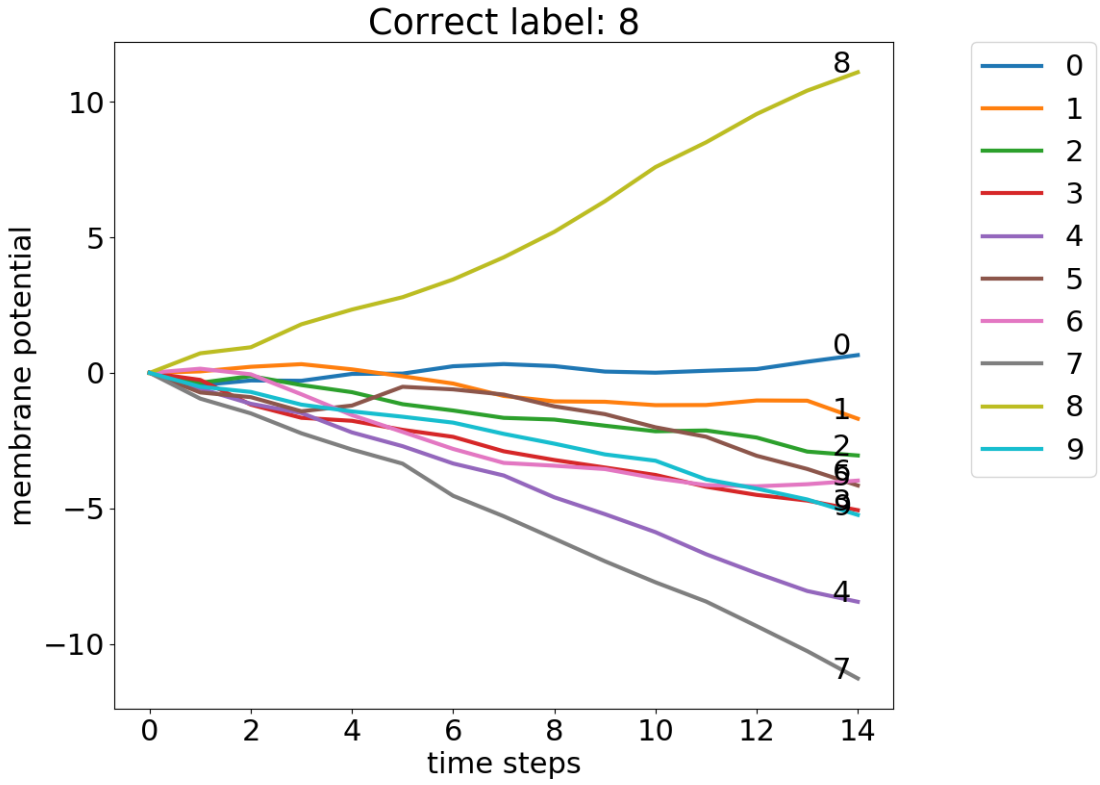


Figure 5.12: Change of the output neuron's membrane potential over time

Comparing the plots in figure 5.12 and figure 5.13 against the evaluation of the membrane potential of the output neurons in the TSCNN model, it can be observed that the decision process of the WSCNN model looks less noisy.

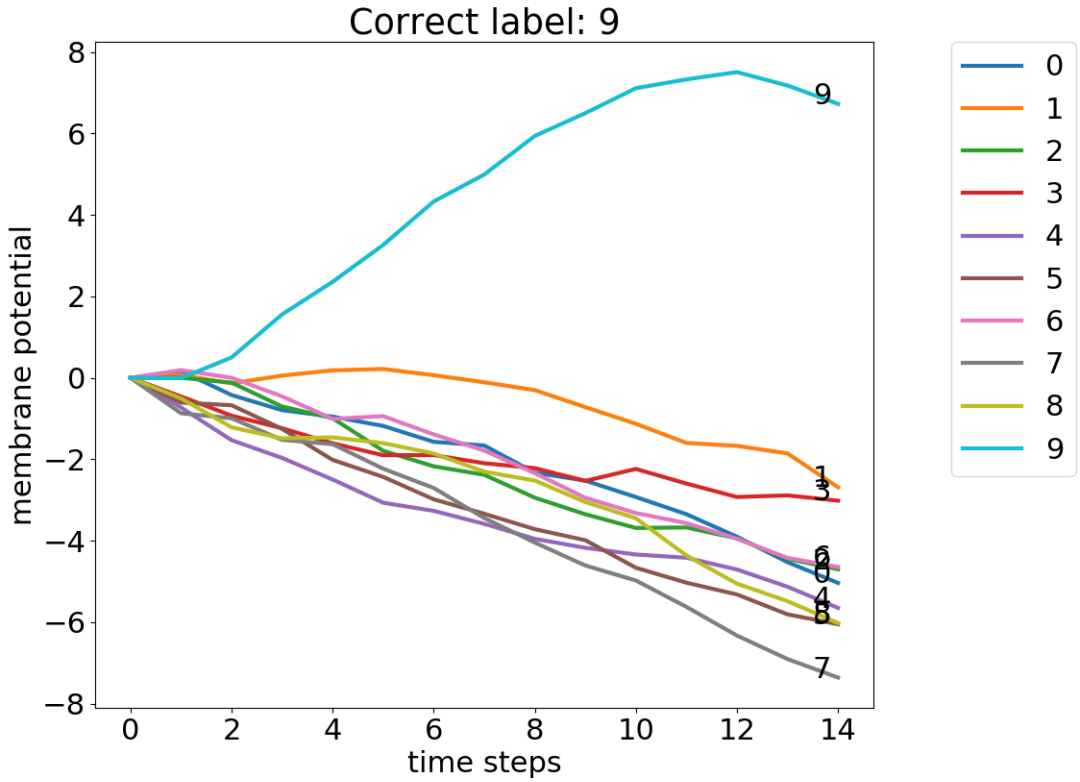


Figure 5.13: Change of the output neuron's membrane potential over time

The difference in the membrane potentials of the output neurons also shows the confidence with which the model made the classification decision. If the membrane potential of one class is clearly far above the membrane potentials of all other neurons, the model was very confident in choosing this class. On the other hand, if the highest membrane potential was not far away from the highest membrane potential of the model's second choice the model is less confident in its decision.
In both figure 5.12 and 5.13, the WSCNN network was very confident in its decision.

Filters

Some of the filters obtained by training the WSCNN can be seen in figure 5.14, 5.15, 5.16. It is important to keep in mind that the depicted filters only represent a small portion of all the filters that are present in the model.

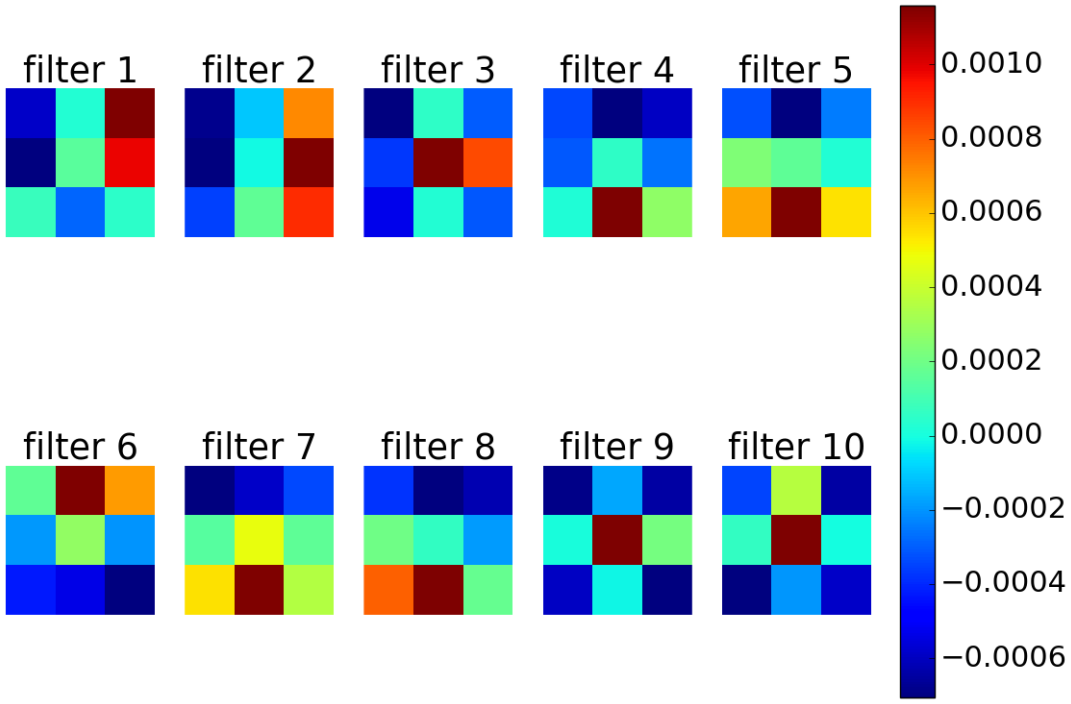


Figure 5.14: Some samples of the trained filters from the first convolutional layer.

The second and the fifth filter in figure 5.14 are fine examples of feature maps, which can pick up on vertical and horizontal lines of input images. As these filters belong to the first convolutional layer they try to find very low-level features in the image.
The filters plotted were randomly selected and not chosen by hand.

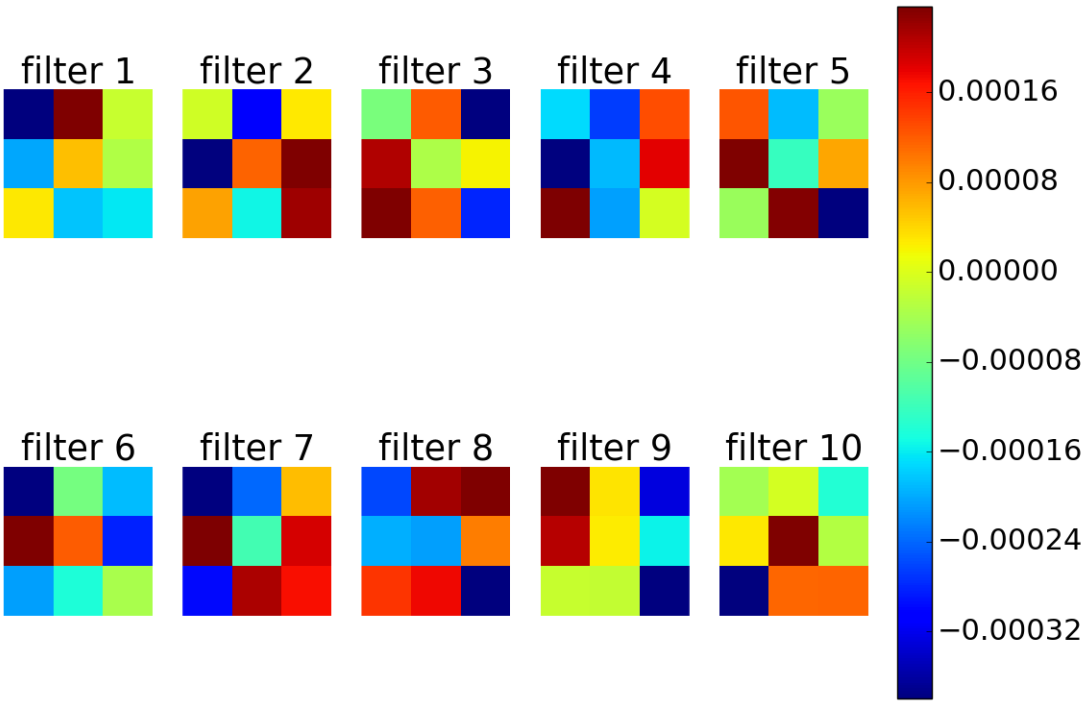


Figure 5.15: Some samples of the trained filters from the second convolutional layer.

The second convolutional layer tries to combine features that have been extracted by the first convolutional layer into new features. The pooling layer additionally shrinks the size of the output of the first convolutional layer, the filters in the second layer try to pick up on more high-level features.

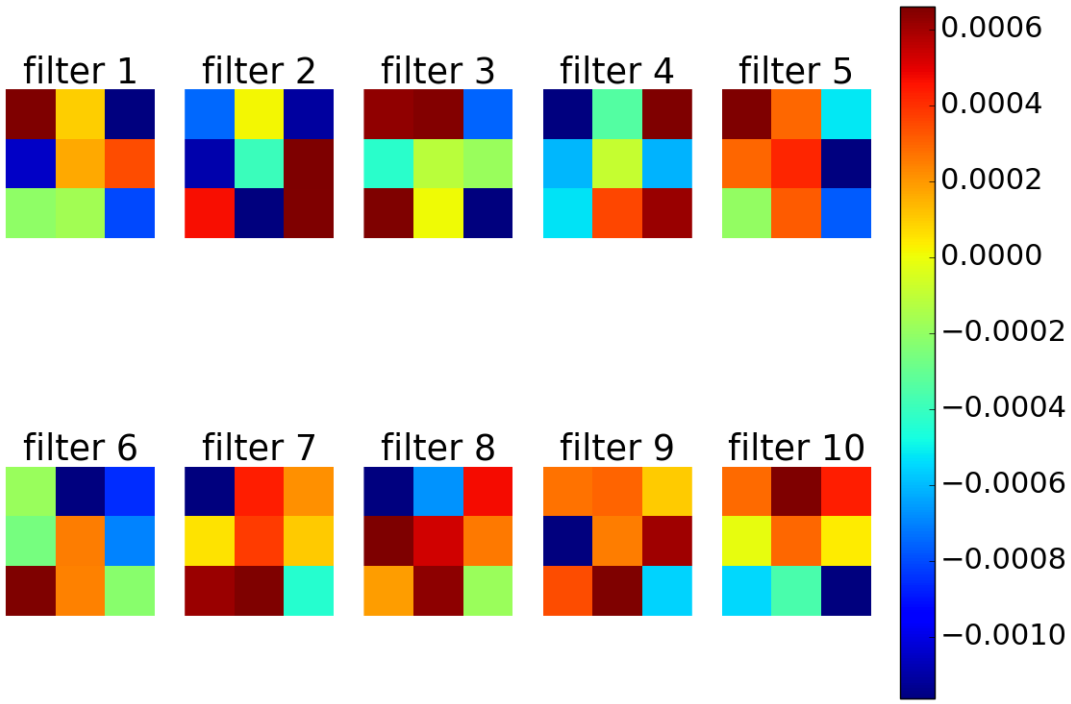


Figure 5.16: Some samples of the trained filters from the third convolutional layer.

Figure 5.16 depicts 10 randomly selected filters used by the third convolutional layer of the WSCNN model.

Confusion matrix

It is also interesting to analyze how the network performs on the individual classes. For example, it is compelling to find out which classes the network confuses most often.
Constructing a confusion matrix is an excellent way of visualizing these relationships. The columns of a confusion matrix correspond to the actual classes and the rows represent the classes that were predicted by the model.

Figure 5.17 shows the confusion matrix of the WSCNN model on the test set. The diagonal of the matrix displays how many samples of this class have been correctly classified. The confusion matrix indicates, that the WSCNN is more likely to mix up different animals than it is to confuse an animal for a mode of transportation. The two classes which seem to be most difficult to distinguish for the model are cats and dogs.

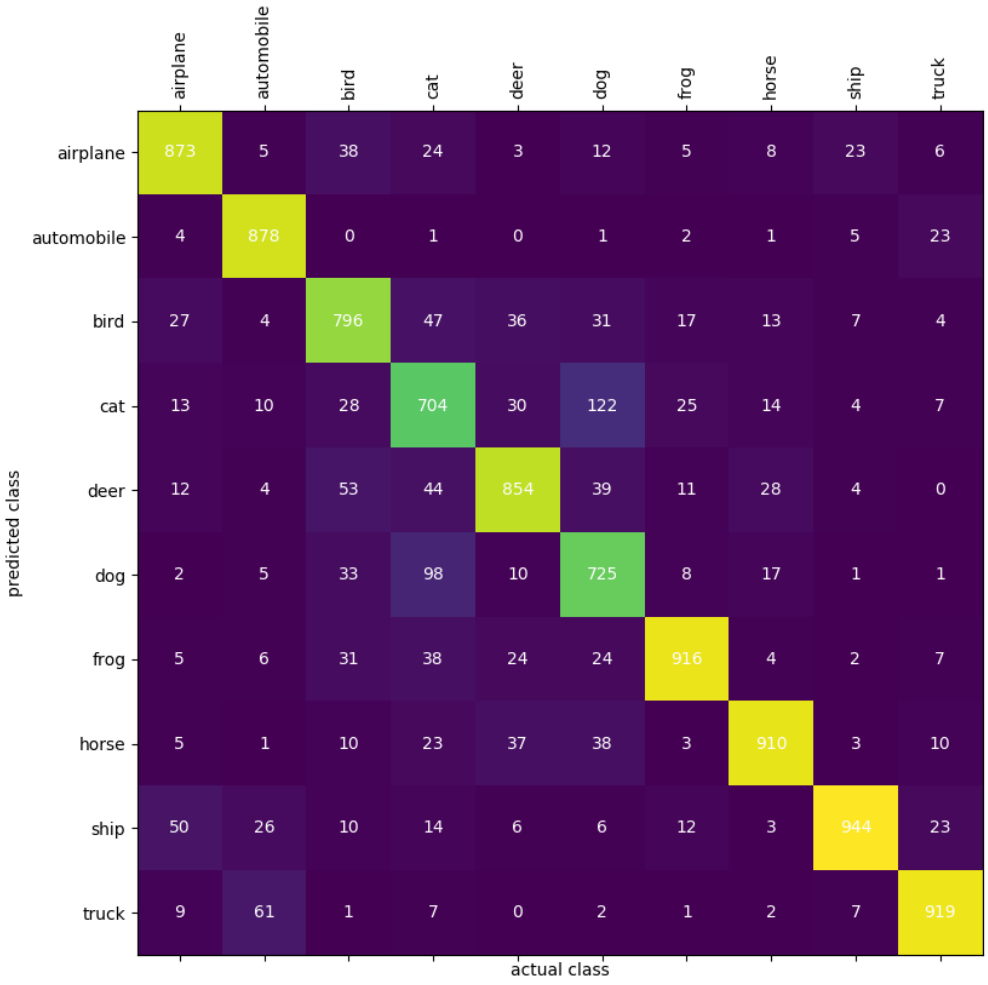


Figure 5.17: Confusion matrix of the WSCNN model.

5.3 Residual SCNN

There is a tendency for CNNs that an increased number of layers has a positive impact on the performance of the network. There are reports (He et al., 2015) that propose architectures which contain more than 100 convolutional layers.
Although depth seems to be a key feature for increasing the performance of CNNs consisting of artificial neurons, most spiking convolutional neural networks reported so far are comparably shallow. For SCNNs the number of layers is usually a single digit.

5.3.1 Architecture

Inspired by the CNN architecture for the CIFAR10 data set from the paper introducing residual connections, (He et al., 2015) RSCNN has been designed to be a considerably deeper spiking convolutional neural network. This SCNN is to the best of my knowledge the deepest SCNN ever reported.

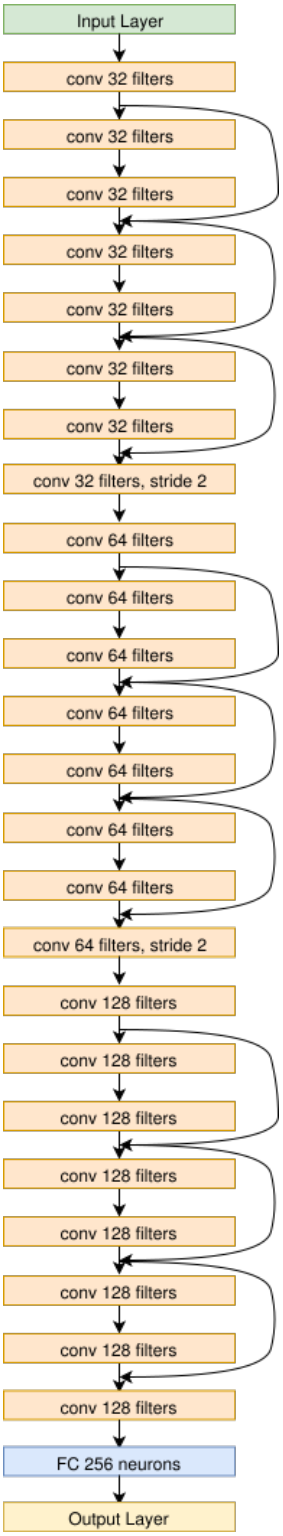


Figure 5.18: Architecture of the RSCNN

Figure 5.18 shows the architecture of the RSCNN model. It consists of 27 layers in total. The convolutional layers, which make up most of the model, have been colored in orange. All filters used in the convolutions have a size of three by three pixel. Towards the end of the network, a fully connected layer (FC) consisting of 256 neurons can be found.
A key feature of this architecture is the usage of residual connections, which are represented by the arrows which skip some layers.
It should be noted, that this network does not include any pooling layers. Instead, there are two convolutional layers utilizing a stride of two.
Although the proposed model seems rather large at first glance, it has considerably fewer parameters than the WSCNN. Information regarding the size of this model can be found in table 5.7.

Number of neurons	462.090
Number of weights	3.648.864

Table 5.7: Number of neurons and weights in the RSCNN model

5.3.2 Input and output convention

The input and output conventions used are identical to the conventions described for the WSCNN model.

5.3.3 Training and Hyperparameters

An extensive hyperparameter search has been carried out to find a set of hyperparameters, which maximizes the test accuracy of the model.
Finding the best set of hyperparameters for a given model is a non-trivial task. Searching for hyperparameters comes which considerable computational costs, as the model has to be trained from scratch for every new hyperparameter configuration.
The values of the hyperparameters listed in table 5.8 were obtained by assuming that the individual hyperparameters can be optimized independently from one another.

batch size	64
learning rate	0.00001
learning rate decay	0.99
learning rate decay exp	0.5
num time steps	10
thr	0.01
decay	0.45
stddev	0.001
mean	0
reg coeff	10
reg tol	0.01
reg shift	-0.008
stretch	1.05

Table 5.8: Hyperparameters used for the RSCNN model

A detailed description of the individual hyperparameters can be found in the appendix.

The regularization scheme used for the WSCNN has also been used for training this model. However, it was slightly adapted to allow for neurons to become more hyperpolarized. This grants neurons to have a more negative membrane potential, strongly inhibiting any spiking activity.

Also, the image preprocessing schemes used for the WSCNN model have been used for training the RSCNN.

5.3.4 Spike plots

In order to gain an insight into the internal spiking activity of the RSCNN model the spikes of 50 randomly selected neurons have been plotted in figure 5.19 and figure 5.20.
As it is infeasible to plot sample neurons from all layers in the RSCNN, only a few layers have been selected. It should be noted, that the spiking activity looks very similar across all convolutional layers in this model.

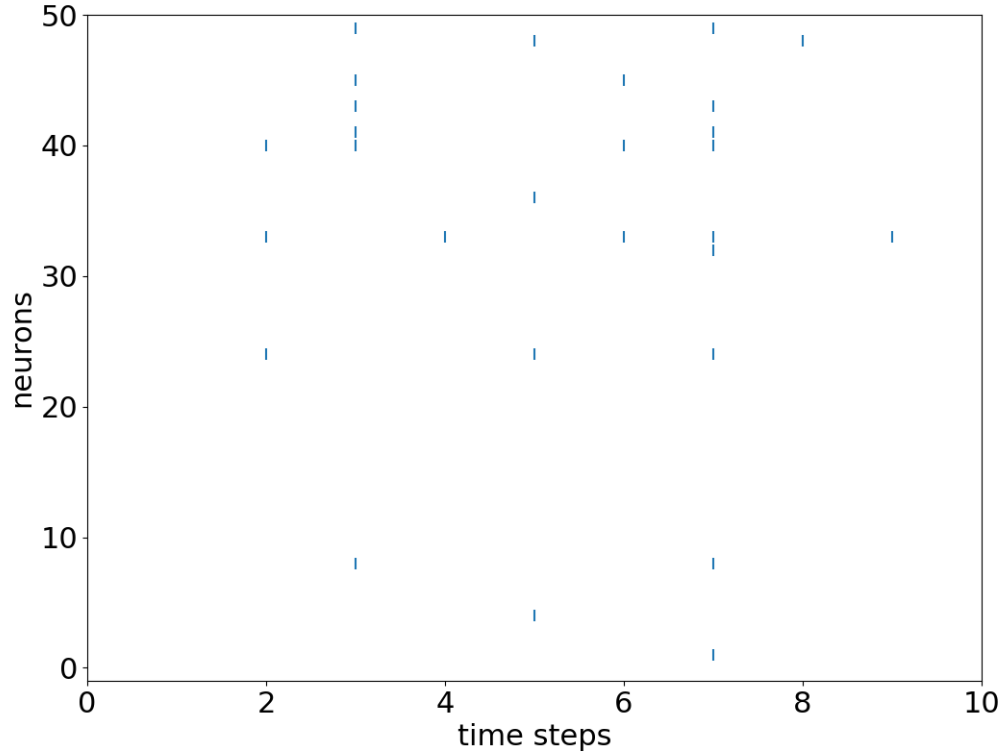


Figure 5.19: Spikes in the eights convolutional layer

In general, the model shows a very sparse spiking activity throughout the convolutional layers. It can be observed that the neurons in the fully connected layer tend to spike more often, as can be discovered by comparing figure 5.19 and figure 5.20.
The majority of the neurons (99.9%) are part of the convolutional layers, therefore it can be concluded that the overall spiking activity is very sparse.

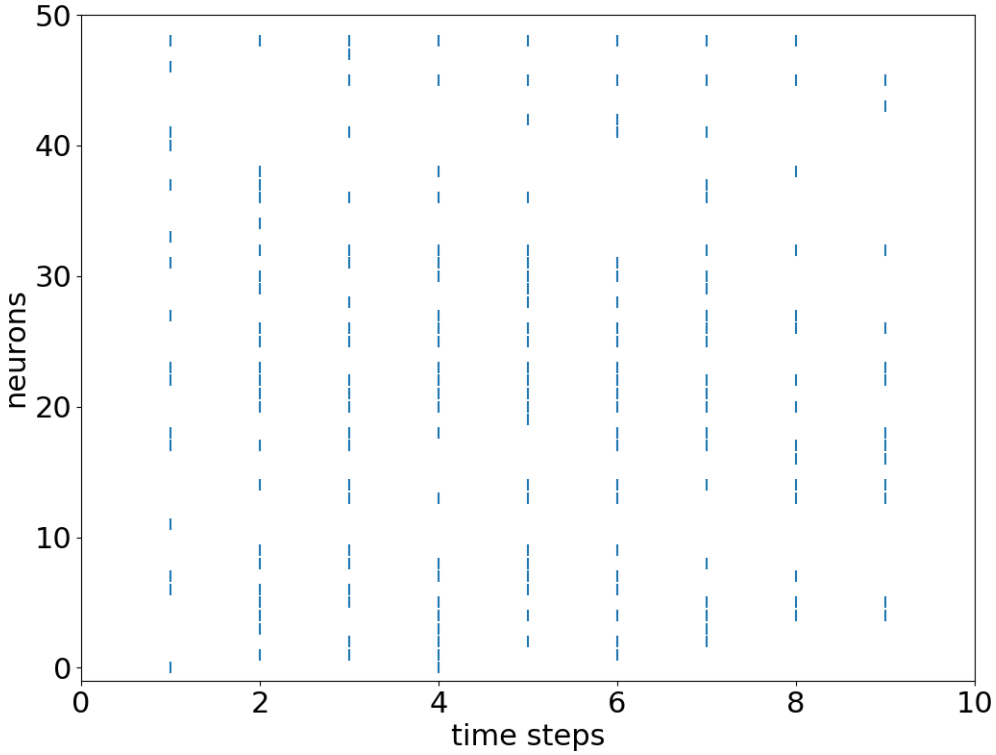


Figure 5.20: Spikes in the fully connected layer

5.3.5 Results

The model manages to achieve a test accuracy of 81.14%.
Figure 5.21 and 5.22 show the evolution of the membrane potentials of the output neurons over time. Both input images have been classified correctly.

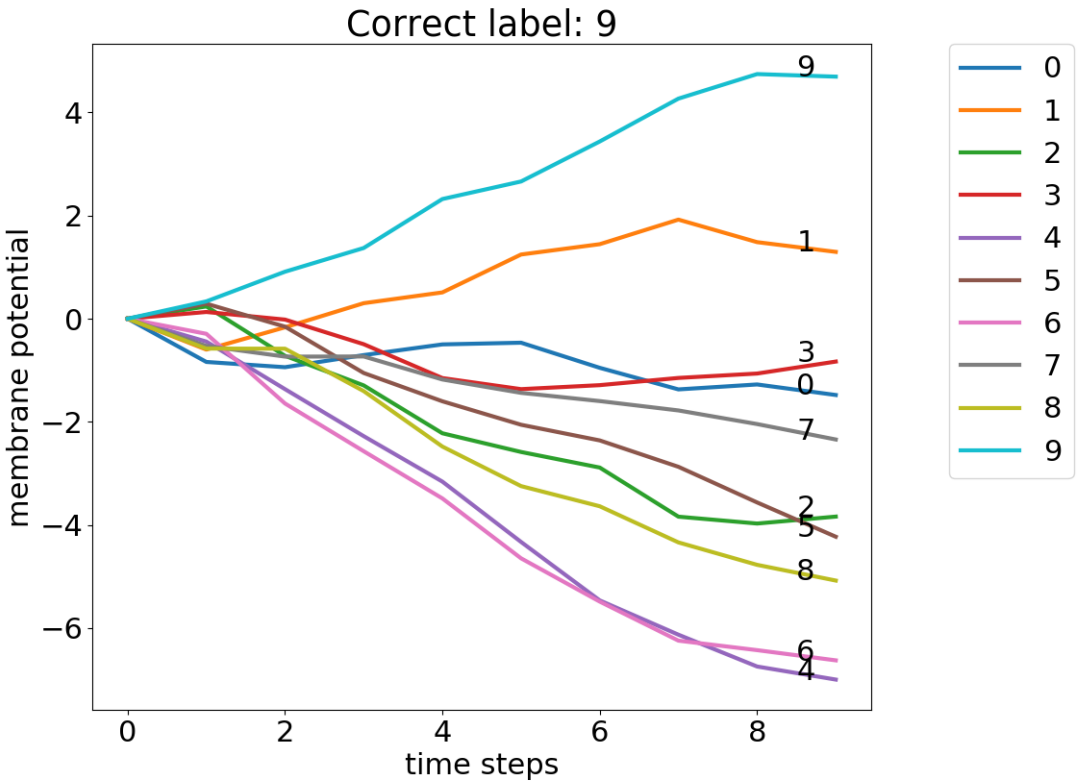


Figure 5.21: Membrane potentials of the output neurons over time.

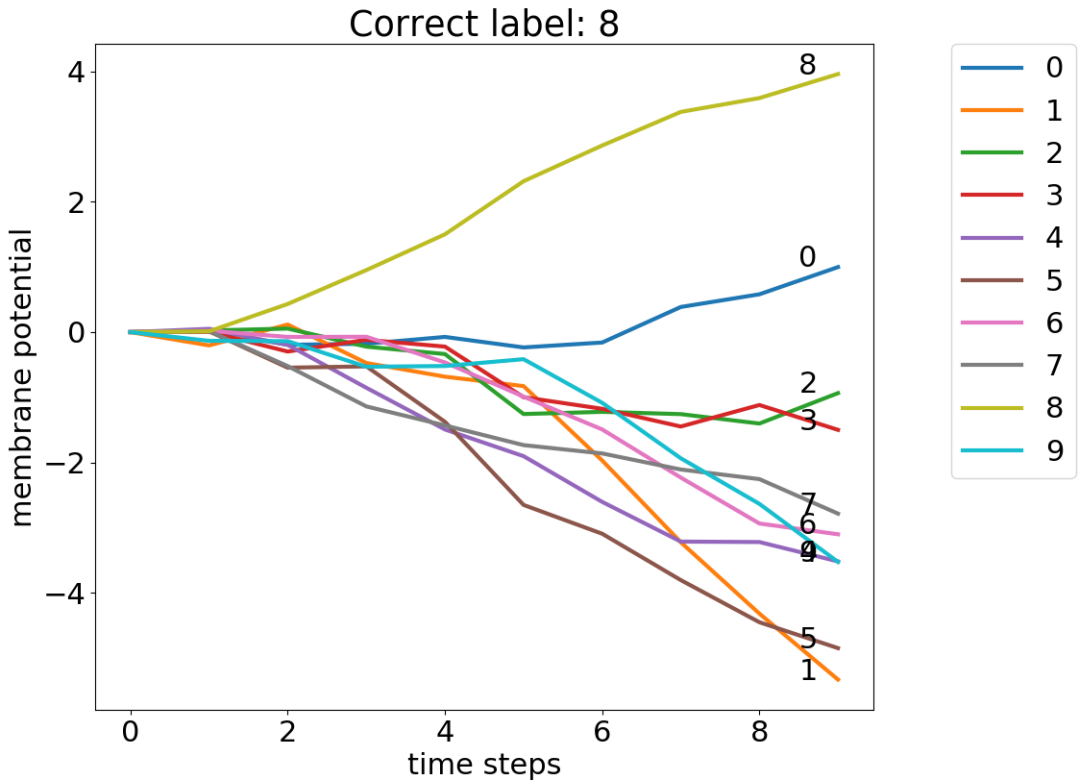


Figure 5.22: Membrane potentials of the output neurons over time.

The change in the output neuron's membrane potential in the WSCNN and in the RSCNN looks comparable. It can be described as less noisy compared to the TSCNN model. Figures 5.21 and 5.22 also suggest that the model mainly increases its confidence in the classification decision over the last time steps.

Confusion matrix

Figure 5.23 shows the confusion matrix of the RSCNN model on the test set. When comparing it to the confusion matrix obtained from the WSCNN model (figure 5.17) it becomes apparent that both networks seem to struggle more with differentiating between images that look more alike. For example,

both networks are more likely to wrongly classify an animal for a different animal. The two classes which were mixed up most frequently are cats and dogs for both the RSCNN and the WSCNN.

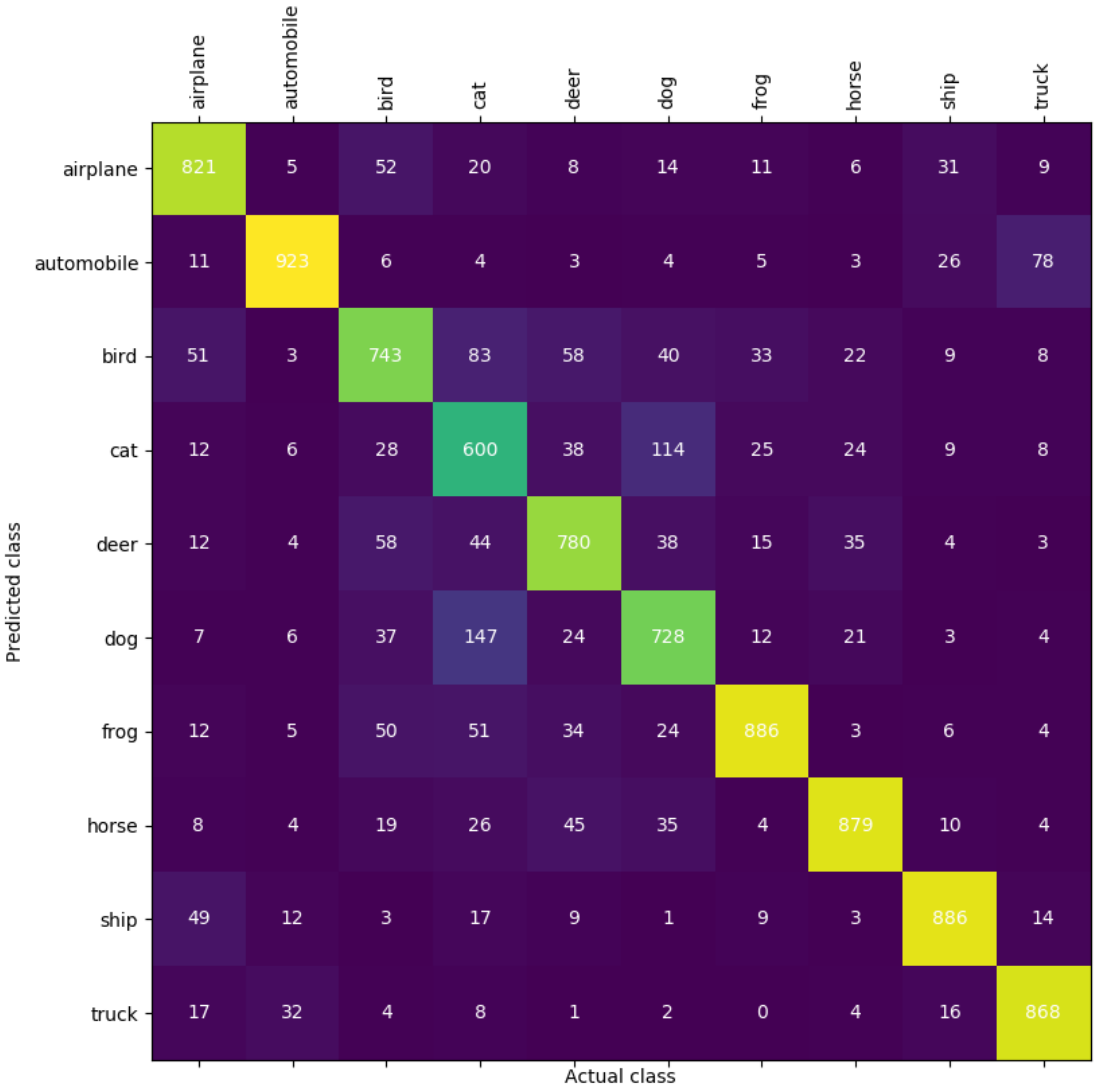


Figure 5.23: Confusion matrix of the RSCNN model.

6 Summary

6.1 Discussion

The TSCNN model demonstrates, that by using the temporal domain it is possible to achieve 89% of the human performance. The proposed input convention, which uses spike timings to encode pixel values, has so far not received a lot of attention in the research community. The more common way of encoding input images is by using a firing rate based scheme. However, these findings indicate that the temporal domain can also be considered as an alternative.
It is also noteworthy that this performance is achieved by a very lightweight neural network consisting of less than 2000 neurons.
The WSCNN model is very different compared to the TSCNN in many aspects. It was designed to be considerably larger, motivated by the goal of increasing performance. With a test accuracy of 85.19%, it is comparable with similarly sized state of the art spiking convolutional neural networks. Furthermore, the WSCNN confirms that large spiking neural networks can be successfully trained using backpropagation through time. Using additional tricks like a good regularization scheme and correctly applied dropouts can increase performance even more.
As there is a strong tendency of CNNs to use an increasing amount of layers it is interesting to investigate how spiking CNNs behave if their depth is increased. The RSCNN is to the best of my knowledge the deepest reported spiking convolutional neural network with a total of 27 layers. The number of weights used in the RSCNN is smaller than the number of weights in the WSCNN model, which might be a reason why the WSCNN has a slightly higher performance.
In addition to being very deep, the RSCNN also has the advantage of not containing any pooling layers. Pooling layers are questionable, as they perform a biologically implausible operation. They also pose a challenge for

possible hardware realizations of the network.
Despite the fact that the RSCNN did not achieve the best performance, deep spiking convolutional neural networks promise to be an interesting research direction. Due to the nature of the convolution, it is easier to scale up CNNs by adding more layers rather than by adding more filters.
As CNNs using artificial neurons have been using more than 100 (He et al., 2015) layers with outstanding results, it makes sense to assume that spiking CNNs could also profit from having more layers.
There are numerous methods that can help to improve the performance of the model. Using a regularization scheme is very important, as it enables the network to receive rich weight updates even after many training iterations. To reduce overfitting it can be beneficial to use dropouts. The image preprocessing schemes described in this master thesis can help to boost the test performance even further. Using a decaying learning rate has also proven to be valuable.

6.2 Conclusion

The Achilles heel of spiking neural networks is still performance, but it seems like the gap between spiking neural networks and artificial networks is growing smaller.
The main goal of this technology is not to beat artificial neural networks, but to approach their performance close enough that the energy efficiency benefits become more and more attractive. This would make spiking neural networks a very appealing solution to a vast number of AI-related problems.
Looking into the future, it will certainly be of interest to extend SCNNs in a way to make them compatible with video data.
This would effectively mean adding an additional dimension (time) to the input data. Fortunately, time is something that spiking neural networks should be able to deal with very naturally.
Porting the models to Loihi would also be an interesting goal, as this would show that highly energy-efficient image classification is possible. Offline image classification tasks on battery-powered devices could be solved this way.

7 Appendix

7.1 Parameter description

This list describes the meaning of the various parameters used in table 5.3.

1. generations: the number of generations the model is trained for.
2. reg coeff: A coefficient which can be used to tune the importance of the regularization term in the loss function.
3. reg tol: The tolerance of the regularization term. This refers to half of the distance of the flat part of the regularization function.
4. reg shift: It is not always ideal to use the threshold as the target value of the regularization function. It can be beneficial to shift the target membrane potential below the threshold as this will allow for more hyperpolarized neurons, which can be beneficial for the computations.
5. batch size: The number of images used in one training step
6. learning rate: The learning rate used by the optimizer.
7. n refractory: The minimum amount of time steps that have to pass until a neuron can fire again.
8. num time steps: The amount of time steps the model is evaluated for.
9. thr: The value of the threshold of the spiking neurons. Once the threshold is surpassed, the neuron will fire.
10. mean: The weights of the model are initialized using a truncated normal distribution. This parameter can be used to set the distributions mean.
11. stddev: The standard deviation of the truncated normal distribution used for initializing the weights.
12. decay: The amount of membrane potential which will be kept from the previous time step.
13. dropout probability: the probability with which a single weight is zero for a given training iteration. Using dropouts is a common technique to avoid overfitting.

14. input noise stddev: Another approach of dealing with overfitting is to add random noise to the training data. This way the network will never see exactly the same image twice and it is forced into a better regime of generalization.
15. stretch: How much the triangle-shaped artificial gradient of the loss function is being stretched on the x-axis. A value above 1 will lead to more non-negative gradients, but also represent a less precise approximation of the actual gradient of the spike function.

7.2 Implementation details

All three models have been implemented and trained using Google's machine learning library TensorFlow (Abadi et al., 2016). This library enables the user to create complex computational graphs which can be run on CPUs, GPUs and also on specialized tensor processing units (TPU). One of the biggest advantages of using TensorFlow is that the framework takes care of the gradient computations.
TensorFlow can be used in C++ and also in python, however the implementation code for this master thesis was just written in python.

Bibliography

Abadi, Martin et al. (2016). "TensorFlow: A system for large-scale machine learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf (cit. on p. 64).

Bellec, Guillaume et al. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons." In: *Advances in Neural Information Processing Systems*, pp. 787–797 (cit. on p. 29).

Beyeler, Michael, Nikil D Dutt, and Jeffrey L Krichmar (2013). "Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule." In: *Neural Networks* 48, pp. 109–124 (cit. on p. 24).

Cassidy, Andrew S. et al. (2016). "TrueNorth: A High-Performance, Low-Power Neurosynaptic Processor for Multi-Sensory Perception, Action, and Cognition." In: (cit. on p. 16).

Chrabaszcz, Patryk, Ilya Loshchilov, and Frank Hutter (2017). "A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets." In: *CoRR* abs/1707.08819. arXiv: 1707.08819. URL: http://arxiv.org/abs/1707.08819 (cit. on p. 4).

Courbariaux, Matthieu et al. (2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." In: *arXiv preprint arXiv:1602.02830* (cit. on p. 15).

Davies, M. et al. (2018). "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning." In: *IEEE Micro* 38.1, pp. 82–99. ISSN: 0272-1732. DOI: 10.1109/MM.2018.112130359 (cit. on pp. 1, 17).

Essera, Steven K et al. (2016). "Convolutional networks for fast energy-efficient neuromorphic computing." In: *Proc. Nat. Acad. Sci. USA* 113.41, pp. 11441–11446 (cit. on p. 15).

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 8, 26, 52, 62).

Hodgkin, Alan L and Andrew F Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of physiology* 117.4, pp. 500–544 (cit. on p. 14).

Ho-Phuoc, Tien (2018). "CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans." In: *CoRR* abs/1811.07270. arXiv: 1811.07270. URL: http://arxiv.org/abs/1811.07270 (cit. on p. 1).

Hu, Yangfan et al. (2018). "Spiking Deep Residual Network." In: *CoRR* abs/1805.01352. arXiv: 1805.01352. URL: http://arxiv.org/abs/1805.01352 (cit. on p. 23).

Huang, Yanping et al. (2018). "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism." In: *CoRR* abs/1811.06965. arXiv: 1811.06965. URL: http://arxiv.org/abs/1811.06965 (cit. on p. 19).

Kheradpisheh, Saeed Reza, Mohammad Ganjtabesh, and Timothee Masquelier (2016). "Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition." In: *Neurocomputing* 205, pp. 382–392 (cit. on p. 24).

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 29, 39).

Krizhevsky, Alex (2012). "Learning Multiple Layers of Features from Tiny Images." In: *University of Toronto* (cit. on p. 5).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 8).

LeCun, Yann et al. (1999). "Object recognition with gradient-based learning." In: *Shape, contour and grouping in computer vision*. Springer, pp. 319–345 (cit. on p. 8).

Lee, Chankyu, Syed Shakib Sarwar, and Kaushik Roy (2019). "Enabling Spike-based Backpropagation in State-of-the-art Deep Neural Network Architectures." In: *arXiv preprint arXiv:1903.06379* (cit. on p. 42).

Masquelier, Timothee and Simon J. Thorpe (2005). "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity." In: *PLoS Computational Biology* 3, pp. 1762–1776 (cit. on p. 24).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning Representations by Back-propagating Errors." In: *Nature* 323.6088, pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0 (cit. on p. 15).

Russakovsky, Olga et al. (2014). "ImageNet Large Scale Visual Recognition Challenge." In: *CoRR* abs/1409.0575. arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575 (cit. on p. 6).

Sengupta, Abhronil et al. (2018). "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures." In: *CoRR* abs/1802.02627. arXiv: 1802.02627. URL: http://arxiv.org/abs/1802.02627 (cit. on p. 23).

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1, pp. 1929–1958 (cit. on p. 42).

Szegedy, Christian et al. (2015). "Rethinking the Inception Architecture for Computer Vision." In: *CoRR* abs/1512.00567. arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567 (cit. on p. 8).

Tavanaei, Amirhossein and Anthony S Maida (2017). "Multi-layer unsupervised learning in a spiking convolutional neural network." In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 2023–2030 (cit. on p. 24).

Wu, Yujie et al. (2018a). "Direct Training for Spiking Neural Networks: Faster, Larger, Better." In: *CoRR* abs/1809.05793. arXiv: 1809.05793. URL: http://arxiv.org/abs/1809.05793 (cit. on pp. 21, 42).

Wu, Yujie et al. (2018b). "Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks." In: *Frontiers in Neuroscience* 12, p. 331. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00331. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00331 (cit. on pp. 16, 19).

Wysoski, Simei Gomes, Lubica Benuskova, and Nikola Kasabov (2008). "Fast and adaptive network of spiking neurons for multi-view visual pattern recognition." In: *Neurocomputing* 71.13-15, pp. 2563–2575 (cit. on p. 24).