MEng Individual Project

Imperial College London

Department of Electrical and Electronic Engineering

Data-driven Classification Methods for Event-driven Cameras

Author:
Tejas Dandawate

Supervisor:
Prof. Pier Luigi Dragotti

Second Marker:
Prof. Patrick A. Naylor

June 21, 2022

Abstract

Neuromorphic sensing is a novel way of encoding analogue signals, inspired by the biological processing of information in our brains. Neuromorphic sensing is based on time encoding, where rather than recording the amplitude of the input signal at predefined times, one records the time instants where the amplitude surpasses a certain trigger mark. This leads to a very efficient way of acquiring and processing signals.

The main aim of the project is to utilise data retrieved from neuromorphic cameras to carry out video classification. Two pipelines are devised to achieve this; one involves directly classifying event data, while the other is a two-phase network were events are first reconstructed into intensity videos before classification is done. With both pipelines, a series of networks are used for classification in order to ensure there is a fair comparison.

Data pre-processing is also a large part of the machine learning pipeline, and as such played a part in the creation of the two pipelines. For the direct-event classification pipeline in particular, varying pre-processing methods are used. In most neuromorphic data analyses a method called 'frame integration' is utilised, where events are segmented into time bins and accumulated into frames. In the classical approach, each frame has two channels depicting whether a positive of negative event occurred in each pixel during a particular time bin. In this project, the classical implementation was used as well as a new novel approach. Multiple networks were designed and implemented to be a part of both classification pipelines. The main ones were convolutional and recurrent neural networks. Finally, a fully neuromorphic system using a spiking neural network is devised, allowing for the exploitation of the sparse data communication between layers. With similar performance, these networks can be utilised for low-powered devices and real-time classification in future works.

Acknowledgements

Thanks mum!

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Neuromorphic data is obtained from event-based cameras that differ from traditional frame-based cameras in that each pixel asynchronously record 'events', which are categorised as large shifts in light intensity. This is a novel representation of data that has overarching benefits that are yet to be fully explored in a plethora of applications. Event-based cameras have the potential to revolutionise efficient, low-power computer vision due to their efficient encoding of information. Since the cameras are asynchronous in nature they allow for low-latency feeds with very little motion blur and other similar visual artefacts. As well as this their pixel structure allows for a very high dynamic range, meaning they can be used even when bright sunlight and dark shadows are encompassed in the same scene.

With the emergence of the Internet of Things (IoT), sensors have become, or at the very least will become, ubiquitous in everyday life. These devices operate using power at a premium since they need to perform adequately with a limited power supply (e.g., small battery cells etc.). This scarcity is further emphasised when attempting to do more power-intensive tasks. One such task is computer vision, which is becoming evermore prevalent as a means of human-computer interaction. Classical frame-based cameras are power intensive, and do not allow for a low active duty cycle (i.e., allowing for device sleep/idle time). The usual way in which this issue is alleviated is by making the system react to an event trigger by 'waking up' to work for a short period of time. Such events may include things such as; motion, timing, acceleration, or temperature. When compared to the function of event-based cameras, where event detection is built into the system, this can be seen as a stop-gap measure. To further leverage the low power-consumption of event-based cameras, spiking neural networks could also be implemented to create a fully neuromorphic pipeline. The added benefit of such a system would be the ability to utilise the sparse data communication between spiking neurons, allowing for even greater power-efficiency for low-powered devices and real-time data analysis.

The high temporal resolution and dynamic range of event-based cameras also presents further benefits when compared to frame-based cameras. These features allow for videos to be represented in a very high fidelity format that preserves more data in fast-paced and brightly lit environments. This may lead to more reliable use of the data for computer vision and even other purposes where the exact environment the camera is set up in is uncertain and unpredictable.

1.2 Problem Specification

The problem this report tries to address is the processing of event-based data in order to employ commonly used computer vision algorithms. This is an exploratory venture into the field, and so the subsection of classification problems was chosen.

Most commonly used classification models have been designed with inputs from frame-based cameras in mind. For this reason, a novel approach was designed wherein event streams are first converted into intensity videos using pre-built and trained models such as the E2VID reconstruction model[4], which can then be fed into pre-existing classification models. This two-phase pipeline can then be compared with classification directly on event data.

Additionally, in the field of event analysis, a method known as frame-integration is commonly applied, which allows for the construction of frames from segments of event data. This method allows for the implementation of custom integration techniques, which could improve the performance of the direct event classification pipelines.

Finally, there is growing interest in spiking neural networks, since they have sparse data communications between neurons. This means that they are more suited for efficient data transfer and fast inference than artificial neural networks. A large benefit of frame-based cameras and their data representations is the energy efficiency, lending itself to use in low-powered devices. The investigation of novel spiking neural networks would allow for a fully neuromorphic system, which could be tested in future work for low-power devices and real-time inference.

1.3 Requirements Capture

- Load and prepare neuromorphic datasets for use in the project.
- Process data and set-up pipeline for two-phase intensity reconstruction and classification.
- Process data and set-up pipeline for direct event analysis.
- Create various classification models to act as a basis of comparison between two classification pipelines.
- Create spiking versions of classification models with similar classification accuracies.
- Create evaluation plan with detailed metrics to compare both the performance of the two pipelines, as well as the individual classification models used in each.
- Compare and contract findings to understand best ways to discern most appropriate event-driven analysis method.

1.4 Report Structure

A brief outline of the report is given below:

Background (Chapter 2)

This chapter gives a background to the project subject. There are outlines of previous works in the field in order to highlight gaps in knowledge where more work can be done. It provides a good illustration of what issues there are and what value there would be in solving them.

Analysis and Design (Chapter 3)

In this chapter a high level overview of the design is given. This includes the choices of hardware/software, as well as architectural diagrams and descriptions.

Additionally this chapter presents a variety of ways in which the final project was evaluated. This allows for a comparison between existing solutions as well every iteration of solution created during the project.

Implementation (Chapter 4)

This chapter outlines noteworthy aspects of the implementation process, as well as a rationale for each step of the projects creation, including any reasons it deviated from the initial design.

Testing and Results (Chapter 5)

Following the evaluation plan in Chapter 3, this chapter aims to apply the planned evaluation techniques to the performance of each of the pipelines and constituent networks. As well as this it includes reasoning for any interesting findings, acting as a basis for the overall evaluation of all implemented systems.

This chapter also includes a critical evaluation of all systems implemented in the duration of the project. An appraisal of both pipelines in general is given, commenting on the effectiveness of event-stream classification. It also acts to measure the success of the project in terms of achieving the aims outlined in Section 1.3.

Finally the success of conversion to spiking networks is also evaluated.

Conclusion and Further Work (Chapter 6)

This final chapter reports the overall success of the project, and outlines what has been achieved in general. As well as this the limits of the project are discussed, including further works that could act as an extension of the project to improve its effectiveness. Finally it includes a comment on the work from others in this field, including any similarities that have arisen between the work created in parallel to this project.

Appendices Appendices including code-snippets, model architectures and details results tables.

Chapter 2

Background

This chapter outlines background information required for understanding the basis for the project. The theory and literature serves to outline the main concepts used for neuromorphic data processing, as well as to reveal gaps in existing research that require solidifying.

2.1 Event Cameras

Event based cameras can be described as 'bio-inspired sensors that differ from conventional frame cameras: Instead of capturing images at a fixed rate, they asynchronously measure per-pixel brightness changes, and output a stream of events that encode the time, location and sign of the brightness changes' [1].

2.1.1 Benefits

Event-based cameras are purported to provide a number of benefits including;

- **High temporal resolution**

  The reason for this is that whereas frame-based cameras have a certain frame-rate, event-based cameras do not have this limitation, meaning the "blind time" between frames is eliminated. The reason for this is that the function of a frame-based camera is dependent on the global shutter to capture the light at a particular instant, whereas event-based cameras can be thought of as having individual shutters for each pixel that are shut whenever an event occurs.

- **High dynamic range**

  The reason for this is again the fact that each pixel has its own individual shutter, but as well as this they all use a logarithmic scale, meaning they function well from very bright to very dim environments as well as fast shifts between the two.

- **Low power consumption**

- **High pixel bandwidth**

  Each pixel can capture events at the rate of kHz. This has the effect of reducing blur since there is a very high temporal resolution to begin with. This makes the system very responsive and therefore ideal for real-time systems.

- **Efficient Encoding**

  Since events are asynchronous and spatially sparse (i.e there are mainly 0 values in the output matrix), the encoding is very efficient, as opposed to frame-based cameras that produce data that is very spatially dense.

The above benefits are very persuasive reasons to adopt neuromorphic cameras in many different applications. It is conceivable that if algorithms can make use of these benefits (since most classical algorithms play to the strengths of the data generated by frame-based cameras), real-time systems could be completely revolutionised.

2.1.2 Function

Event-based cameras differ from frame based cameras fundamentally, in that they do not rely on a global shutter closing at regular intervals to record information of a scene. Instead each pixel closes whenever it detects an 'event' occurring. The way such events are detected is dictated by the 'event generation model'[1].

Each pixel responds to changes in its log photo-current ($L = log(I)$, where I is the perceived brightness), giving the system a very high dynamic range. A recorded event 'k' has the format $e_k = (\mathbf{x}_k, t_k, p_k)$. This is known as the Address Event Representation (AER). The first value is the spacial location of the event ($\mathbf{x}_k = (x_k, y_k)^\top$), the second value t_k is the temporal location, and the final value $p_k \in 1, -1$ indicates the polarity of the event (i.e in which direction the brightness gradient was changing). The brightness increment between two events at the same pixel is given by the equation $\Delta L(\mathbf{x}_k, t_k) = L(\mathbf{x}_k, t_k) - L(\mathbf{x}_k, t_k - \Delta t_k)$. In a perfect (noise free) environment an event is fired whenever the brightness increment reaches a temporal contrast threshold. This relationship is shown in Eq. (2.1).

$$\Delta L(\mathbf{x}_k, t_k) = p_k C (C > 0) \tag{2.1}$$

It should be noted that the value of C could be variable and therefore different for $p_k = \pm 1$. Additionally, we can approximate the temporal derivative of a pixels brightness by substituting Eq. (2.1) into the Taylor expansion given in Eq. (2.2). Eq. (2.3) shows the resulting format for the temporal derivative.

$$\Delta L(\mathbf{x}_k, t_k) \approx \frac{\delta L}{\delta t}(\mathbf{x}_k, t_k)\Delta t_k \tag{2.2}$$

$$\frac{\delta L}{\delta t}(\mathbf{x}_k, t_k) \approx \frac{\Delta L(\mathbf{x}_k, t_k)}{\Delta t_k} = \frac{p_k C}{\Delta t_k} \tag{2.3}$$

It should however be noted that the approximation in Eq. (2.2) is only true under the assumption that Δt_k is exceedingly small. Since unlike frame-based cameras we do not measure absolute brightness, this is an indirect way of measuring and keeping track of the brightness within the frame.

Fig. 2.1 shows the basic functionality of an event-based camera. Shown in **(a)** is the simplified circuit diagram of the DAVIS pixel, which in **(b)** is used to convert light into events (shown in real life in images **(c)** and **(d)**). **(e)** shows how this setup would view a white square rotating on a black disk. It is a stream of events going from the past in green to the present in red. These events can then be seen overlaid on a natural scene in **(f)**.

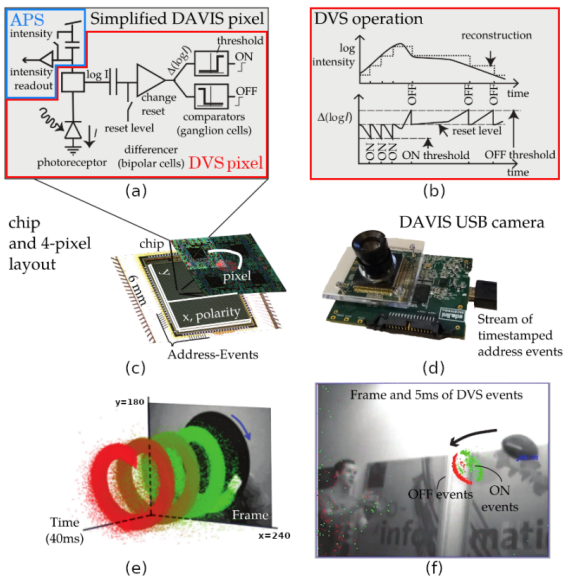


Figure 2.1: A summary of the functionality of the DAVIS event based camera[1].

2.2 Spiking Neural Networks and Neural Heterogeneity

Section 2.1.1 lists some persuasive reasons for utilising neuromorphic systems, but there still many challenges posed when attempting to do so. For example, each pixel only responds to brightness change, but the problem is that such a change could be a result of not only scene changes, but also the position of the camera within the scene. For this reason most neuromorphic systems have currently been limited to stationary cameras. As well as this, the system is especially prone to stochastic noise due to inherent shot noise in photons and from transistor circuit noise [1]. For this reason Eq. (2.1) can only be said to be true under ideal conditions. A more realistic model for events fired is a probabilistic event generation model. These take into consideration the aforementioned sources of noise. One such model is given by P. Lichtsteiner *et al.*[11], where sensor variation measurements suggested a normal distribution centred around C for event triggers.

In order to tackle issues such as the ones due to noise, it is useful to look at existing examples of spiking neural systems, such as a biological brain. It is known that the brain is heterogeneous on every scale, in the past this was thought to be simply a by-product of noisy processes, but more recently it can be shown that by adding heterogeneity to Spiking Neural Networks (SNNs), a more stable and robust system can be created[12], indicating this heterogeneity has a more deep rooted purpose. As well as this the learned neural parameters tend to resemble what can observed experimentally. This may serve as an explanation for how the brain has evolved to deal with the many stochastic processing it encounters.

The foundations of SNNs are in computational neuroscience. The mechanisms of neurons in the brain are the inspiration behind creating ANNs with neurons that spike in the same way. Neurons in an a typical ANN have a weight, bias and activation function. This means that the output of the neurons can be summarised by Eq. (2.4).

$$y = \theta(\sum_{j=1}^{n} w_j x_j - u_j) \tag{2.4}$$

This model was inspired by the biological neuron model shown in Fig. 2.2. The function of the neuron is similar in that it produces an output based on a function of the input stream, but the form of this output is in the form of a 'spike' rather than a continuous function. The Σ shown in the diagram is actually the integration of all excitory and inhibitory input signals to the dendrites coming from the soma of the neuron. We can see that the electric potential of the neuron needs to exceed a certain threshold in order for the output of the neuron to be a spike. Spiking neural networks use a similar model of neurons in order to leverage the aforementioned benefits of neural heterogeneity.

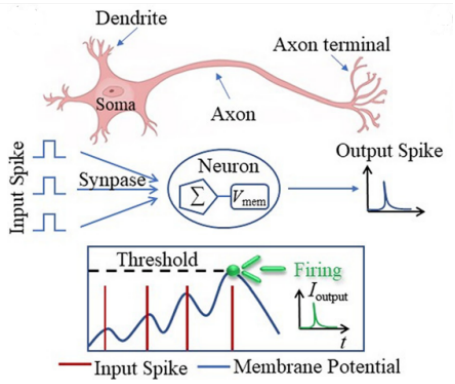


Figure 2.2: A diagram of structure and function of a biological neuron [2].

With SNNs it is theoretically possible to achieve very high energy efficiency, and when combined with a novel surrogate gradient and Recurrent Neural Network (RNN) as was experimented with in a paper by Bojian Yin *et al.*[13] excels in challenging tasks such as gesture recognition, and at some points even outperforms traditional ANNs. Alongside the aforementioned neural heterogeneity to combat stochastic processes we can see substantial improvements in previous SNN performance.

There have been many advancements in the application of deep neural network techniques to SNNs, one of which is the conversion of deep ANNs to SNNs. For example, work by Bodo Rueckauer

et al.[14] out lines an approach to approximate well known ANN layers in a neuromporphic form. With this technique most ANNs can be directly converted to an equivalent SNN. As well as this they implement a technique to train the neural network using existing back-propagation techniques, which was first implemented by Eric Hunsberger *et al*[15].

2.3 Existing Algorithms for Event Analysis

For SLAM, pose estimation and classification tasks the problem again is that classical systems heavily rely on the structure of conventional camera's outputs, and so there needs to be a radical paradigm shift in order to take events as inputs instead.

2.3.1 Optic-flow Methods

Since obtaining the equation for the temporal derivative in Eq. (2.3), there is now a indirect measure of brightness and so a more classic computer vision techniques using optic-flow constraints can be utilised to characterise the events detected by pixels. In frame-based systems, optic flow methods create a flow-field that describe the displacement vector (signifying direction and magnitude of movement) for each pixel in the frame, and a similar derivation can be done for event data. A core constraint in this derivation is that the intensity of a local time-varying image region is constant under motion (for at least a short amount of time)[16]. Eq. (2.5) is the resulting equations that shows the relation between the brightness gradient and the displacement of the pixel over a short period of time given its velocity[1]. It implies that if the motion is parallel to the edge, there is no event fired (since $v \cdot \nabla L = 0$) and conversely if the motion is perpendicular to the edge events are fired at their highest rate.

$$\Delta L \approx -\nabla L \cdot v \Delta t_k \tag{2.5}$$

Optic flow methods serve as a method for finding the magnitude and direction of apparent motion in individual still frames of a video. This is an added dimension of potentially relevant information for classification task where motion are of particular importance (e.g. gesture recognition). For example, a method implemented by Qiguang Miao *et al.*[3] is shown in Fig. 2.3. The network shows multimodal data, including RGB, depth and optical flow data generated from the RGB ones, being sent to the ResC3D network to extract spatio-temporal features. Finally, the features are blended together using canonical correlation analysis, and the final recognition result is obtained by a linear SVM classifier.

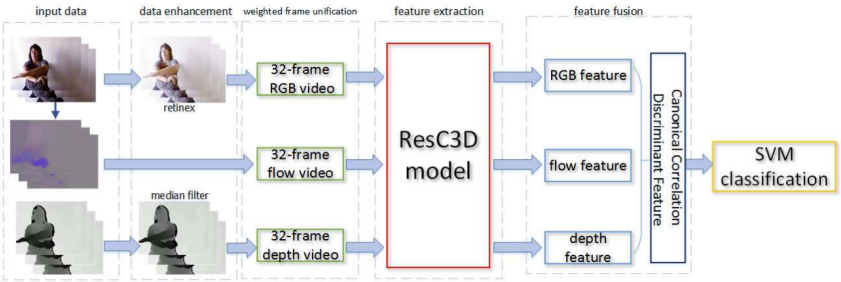


Figure 2.3: A diagram showing the pipeline for a multimodal gesture recognition system[3].

2.3.2 Frame Integration of Event Streams

A common method of getting frame-like videos from event data is to use the event-to-frame integration method. One such method is outlined by *Wei Fang et al.*[17], which is the one used in the python package spikingjelly[18]. The method used is outlined in Eq. (2.6).

Data in neuromorphic datasets are in the formulation of $E(x_i, y_i, t_i, p_i)$ that represent the event's coordinate, time and polarity. We split the event's number N into T slices with nearly the same number of events in each slice and integrate events to frames. Note that T is also the simulating time-step. Denote a two channels frame as $F(j)$ and a pixel at (p, x, y) as $F(j, p, x, y)$, the pixel value is integrated from the events data whose indices are between j_l and j_r:

$$j_l = \lfloor \frac{N}{T} \rfloor \cdot j$$

$$j_r = \begin{cases} \lfloor \frac{N}{T} \rfloor \cdot (j+1), & \text{if j} < \text{T - 1} \\ N, & \text{if j} = \text{T - 1} \end{cases}$$

$$F(j,p,x,y) = \sum_{i=j_l}^{j_r-1} I_{p,x,y}(p_i, x_i, y_i) \tag{2.6}$$

where $\lfloor . \rfloor$ is the floor operation, $I_{p,x,y}(p_i, x_i, y_i)$ is an indicator function and it equals 1 only when $(p, x, y) = (p_i, x_i, y_i)$.

With this formulation, we can obtain a stream of frames which show clusters of events in a more easily processable format. These frames can either capture a number of events in a certain time-slice, in which case the frames will be synchronous in a similar way to the output of classical frame-based cameras, or create time-slices with a set number of events in each. In the latter case the output frames will not be of a constant framerate, and will instead be asynchronous. They do, however, ensure that there is a constant amount of information per frame regardless of the amount of motion at any one time, which may be better for pattern recognition.

2.4 Video Reconstruction Algorithms

Image reconstruction has been implemented for event data building on the direct optimised versions of Convolutional Neural Networks (CNNs). An example of this is the network named 'U-net'[19] which managed to reconstruct a video using 10M parameters to analyse events from an AER camera protocol. Recent work by Rebecq *et al.* illustrates a novel network architecture that reconstructs a video from a stream of events [4]. These methods are purported to allow the introduction of mainstream computer vision research to event cameras. Figure 2.4 shows an example of how converting spiking data to a video stream allows for use of classical computer vision algorithms.

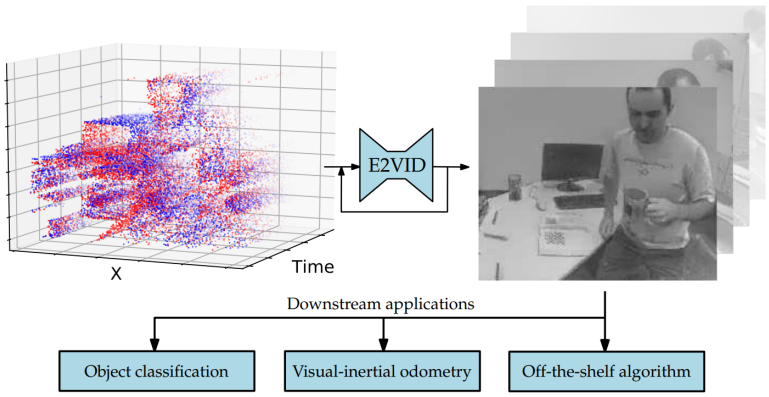


Figure 2.4: An illustration of the mapping of spiking data to video stream to apply off-the-shelf algorithms to[4].

A naive approach would be take each event $e_k = (\mathbf{x}_k, t_l, p_k)$ and assume that the firing was due to a brightness change above a threshold $\pm C$ which is a constant that could be set by the user. If this was the case events could be directly integrated to recover the intensity map of images. however, the value C in reality does not remain constant and is heavily dependent on other factors such as event rate, temperature, and sign of brightness change. The implementation outlined instead makes use of a Recurrent Neural Network (RNN), that takes as input sets of events within a spatio-temporal window. For example, a stream of events will be broken down into sequences given by $\epsilon_i \; \forall i \in [0, N-1]$. Since each sequence is of fixed length N the framerate of the output video from the RNN is proportional to the event rate. Figure 2.5 shows the functionality of such

a network. Each event window ϵ_k is converted to a 3D event tensor and passed into the network along with the last K constructed images to generate the latest iteration of the image. It is clear from this that each new image is constructed by fusing the previous K images with the new stream of events.

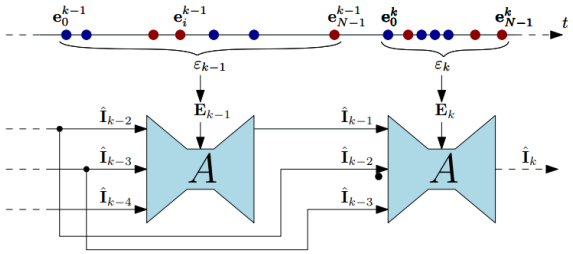


Figure 2.5: An overview of RNN used to generate video from sets of events[4].

2.5 End-to-end Event Classification Networks

A method such as the one described in Section 2.4 allows for the use of hugely researched and well documented computer vision algorithms for classification and other tasks while maintaining the benefits of event-based cameras. However, it may be possible to bypass the intermediate step of video reconstruction altogether, and simply create a model that simply acts as a black box and can be trained to give the required output directly from a neuromorphic input. Figure 2.6 shows how a frame-based video is converted to a continuous stream of events from which snapshots of events can be taken. It should be noted that the distribution of the data from the neuromorphic camera is much more dense than the frame-based video, which means that the motion blue visible in the video should not be a problem with the new representation (as explained in Section 2.1.1).

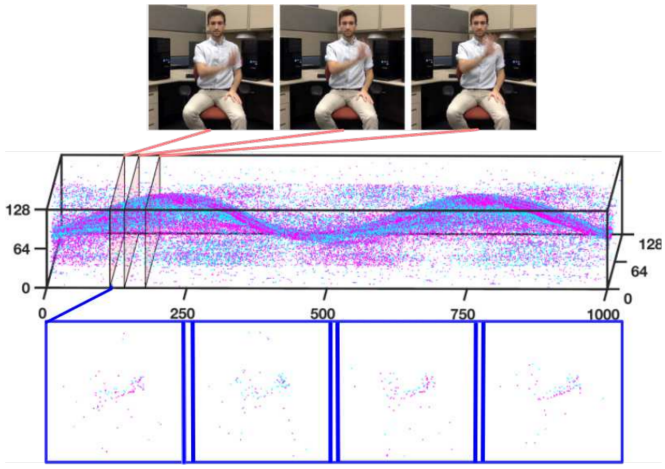


Figure 2.6: An illustration of a temporal filter that caches events fired from a camera[5].

In a paper by Arnon Amir *et al.*[5] a system was created to perform gesture recognition from event-based data. It makes use of the a system such as the one shown in Fig. 2.6 to act as one of a set of temporal filters in a cascade. This cascade feeds into a convolution layer and in the end a winner takes all filter is applied to identify the gesture. The whole network can be seen in Fig. 2.7. The intermediate representations of all the layers can also be seen, showing how important features are being identified similar to how they would have been with traditional frame-based inputs.

There have been other implementations of systems to carry out complex tasks such as classification, and each has a different way of dealing with spiking input data other than with temporal filters. For example a paper by Xavier Lagorce *et al.*[20] moves towards building time surfaces from events to feed into a neural network, whereas Yin Bi *et al.*[21] propose using a non-uniform sampler to create a graph from events to then feed into a network of so-called graph convolution networks.

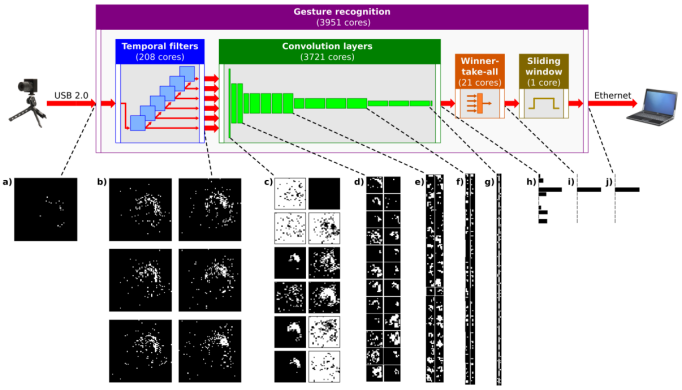


Figure 2.7: A network designed to perform gesture recognition using neuromorphic input by making use of cascading temporal filters[5].

2.6 Temporally Aware Deep Learning and Classification Models

2.6.1 3-Dimensional Convolutional Neural Network

Convolutional neural networks have been shown to be much more effective when processing images that networks built solely with dense, fully-connected layers TODO: add references here . This is because they are able to better identify spacial patterns within an image as a kernel spans more than one pixel. For this reason the basic architecture was to have an input layer (the structure of which is dependent on the input format), followed by a series of hidden convolutional layers of varying parameters. However, since the inputs to systems handling event data are in fact 3D tensors of multiple images (i.e. the integrated frame video generated from the camera events, the process of which is described in Section 2.3.2) a typical convolutional network is not sufficient to capture the temporal patterns in the data. Typically 2D convolution layers can take as input images with three channels (usually RGB), and so feasibly this could be extended to more channels for each frame of the video, but this is not a scalable approach. Instead 3D convolutional layers can be used [22]. For these layers multiple frame time-slices are concatenated into a 3D tensor so that convolutions can occur over both the spacial and temporal dimensions. This allows for patterns to be detected across multiple adjacent frames.

2.6.2 Long-Short Term Memory Networks

When attempting to store information learned in previous frames of a multi-framed input to a network, recurrent back-propagation is a long process due to decaying error backflow. Long-short term memory (LSTM) networks[23] address these issues with efficient, gradient-based methods. These systems were tested to run much more quickly, efficiently and successfully than their recurrent network counterparts. They also allow networks to learn more complex patterns such as long-time-lag tasks that previously plagues ANNs. They are gated networks with separate memory cells. The three gates (input, forget and output) control the flow of information through each cell, acting as filters regulating what information is to be kept and discarded at each iteration.

2.6.3 Gated Recurrent Unit Networks

The recently proposed Gated Recurrent Unit (GRU), is similar in design to the LSTM, since that is also a gated unit. It is more simple than the LSTM, having only two gates in its construction, and omitting separate memory cells. The performance of GRU networks, alongside LSTMs, is shown to be far superior than traditional RNNs[24]. Since it is a more simple design, GRUs have fewer training parameter when compared to LSTMs, and therefore train faster. Though performance is often similar to LSTM networks, it has been noted that on very large datasets they underperform slightly, and in cases where accuracy is concerned they may be less ideal than LSTMs.

2.6.4 Convolutional LTSM Network

LSTM networks show promise in their ability to find patterns not only on an frame-by-frame basis but also in the time dimension. These spatio-temporal patterns are much more effective for classifying image sequences that just spatial patterns since information is carried throughout the frames to find overall movements and gestures. These LTSMs utilise fully connected layers to find patterns in each frame, however it has been shown that convolutional neural networks produce much better results when operating on image data, and so it would stand to reason that LTSMs would benefit from their structure as well. ConvLSTM, developed by *Xingjian SHI et al.*[25], showed great promise, and in their experiments captured spatio-temporal correlations better and more consistently FC-LSTM, outperforming it by a sizeable margin in the application of forecasting.

2.7 Existing Datasets

There already exists many repositories of recorded neuromorphic data to get familiar with spiking data. In this section there are some examples of such datasets and a quick overview of their contents.

2.7.1 Neuromorphic Datasets

The below datasets are created using one of a variety of event-based cameras available on the market. the function of each of the cameras is fundamentally similar (as described in Section 2.1.2) but they also have some differences between them. The cameras widely available today are shown in Fig. 2.8.

	Supplier	iniVation			Prophesee				Samsung			CelePixel		Insightness
	Camera model	DVS128	DAVIS240	DAVIS346	ATIS	Gen3 CD	Gen3 ATIS	Gen 4 CD	DVS-Gen2	DVS-Gen3	DVS-Gen4	CeleX-IV	CeleX-V	Rino 3
Sensor specifications	Year, Reference	2008 [2]	2014 [4]	2017	2011 [3]	2017 [67]	2017 [67]	2020 [68]	2017 [5]	2018 [69]	2020 [39]	2017 [70]	2019 [71]	2018 [72]
	Resolution (pixels)	128 × 128	240 × 180	346 × 260	304 × 240	640 × 480	480 × 360	1280 × 720	640 × 480	640 × 480	1280 × 960	768 × 640	1280 × 800	320 × 262
	Latency (µs)	12μs @ 1klux	12μs @ 1klux	20	3	40 - 200	40 - 200	20 - 150	65 - 410	50	150	10	8	125μs @ 10lux
	Dynamic range (dB)	120	120	120	143	> 120	> 120	> 124	90	90	100	90	120	> 100
	Min. contrast sensitivity (%)	17	11	14.3 - 22.5	13	12	12	11	9	15	20	30	10	15
	Power consumption (mW)	23	5 - 14	10 - 170	50 - 175	36 - 95	25 - 87	32 - 84	27 - 50	40	130	-	400	20-70
	Chip size (mm^2)	6.3 × 6	5 × 5	8 × 6	9.9 × 8.2	9.6 × 7.2	9.6 × 7.2	6.22 × 3.5	8 × 5.8	8 × 5.8	8.4 × 7.6	15.5 × 15.8	14.3 × 11.6	5.3 × 5.3
	Pixel size (µm^2)	40 × 40	18.5 × 18.5	18.5 × 18.5	30 × 30	15 × 15	20 × 20	4.86 × 4.86	9 × 9	9 × 9	4.95 × 4.95	18 × 18	9.8 × 9.8	13 × 13
	Fill factor (%)	8.1	22	22	20	25	20	> 77	11	12	22	8.5	8	22
	Supply voltage (V)	3.3	1.8 & 3.3	1.8 & 3.3	1.8 & 3.3	1.8	1.8	1.1 & 2.5	1.2 & 2.8	1.2 & 2.8		1.8 & 3.3	1.2 & 2.5	1.8 & 3.3
	Stationary noise (ev/pix/s) at 25C	0.05	0.1	0.1	-	0.1	0.1	0.1	0.03	0.03		0.15	0.2	0.1
	CMOS technology (nm)	350	180	180	180	180	180	90	90	90	65/28	180	65	180
		2P4M	1P6M MIM	1P6M MIM	1P6M	1P6M CIS	1P6M CIS	BI CIS	1P5M BSI			1P6M CIS	CIS	1P6M CIS
	Grayscale output	no	yes	yes	yes	no	yes	no	no	no	no	yes	yes	yes
	Grayscale dynamic range (dB)	NA	55	56.7	130	NA	> 100	NA	NA	NA	NA	90	120	50
	Max. frame rate (fps)	NA	35	40	NA	NA	NA	NA	NA	NA	NA	50	100	30
Camera	Max. Bandwidth (Meps)	1	12	12	-	66	66	1066	300	600	1200	200	140	20
	Interface	USB 2	USB 2	USB 3		USB 3	USB 3	USB 3	USB 2	USB 3	USB 3			USB 2
	IMU output	no	1 kHz	1 kHz	no	1 kHz	1 kHz	no	no	1 kHz	no	no	no	1 kHz

Figure 2.8: A table listing widely available event-based cameras and their respective features[1].

NMNIST

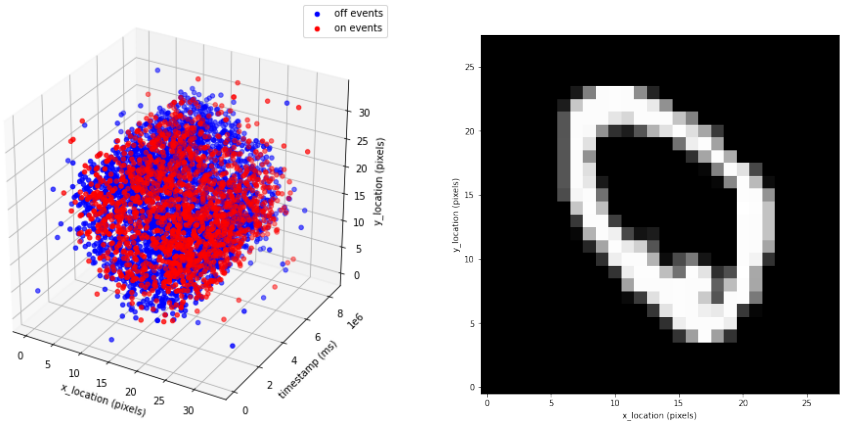
This dataset is a spiking version of the original frame-based MNIST dataset [26][10]. It is identical to the original MNIST dataset, which is a set of handwritten digits, in all ways (including scale, size and sample split) bar one - it was captured using an ATIS sensor mounted on a motorised pan-tilt unit. This sensor moved while viewing the MNIST examples on an external monitor.

For each item in the dataset there is a binary file which has a list of events. Each event is characterised by a 40 bit unsigned integer. The integer gives the following information of a particular event:

- bit 39 - 32: X location (in pixels)
- bit 31 - 24: Y location (in pixels)
- bit 23: Polarity (0 for OFF, 1 for ON)
- bit 22 - 0: Timestamp (in microseconds)

An example of a visualisation of this data is shown in Fig. 2.9, where the image used from the MNIST dataset is on the right in part **(b)** and the resulting spikes from the event-based camera are shown on the left in **(a)**. Here, 'on events' are events where the intensity of a the particular pixel

increased by an increment greater than a threshold, and the 'off events' are when the intensities decrease by a increment greater than a threshold. The representation is clearly very different to the one given by a classical camera, and therefore the use of such data has to have a different approach to classical techniques.



(a) (b)

Figure 2.9: A visualisation of events, **(a)**, from a single training sample from the NMNIST dataset, **(b)**.

DVS128 Gesture

This dataset is a set of 11 hand gestures from 29 subjects under various illumination conditions with a DVS camera. It was created to help create a real-time gesture recognition system that utilises the low power capabilities of event-based cameras[6]. A visualisation of these gestures can be seen in Fig. 2.10. The illumination conditions include; fluorescent led, fluorescent, lab lighting, led lighting and natural lighting. This dataset is ideal for testing a networks capability for identifying spatio-temporal patterns in order to correctly classify each gesture.

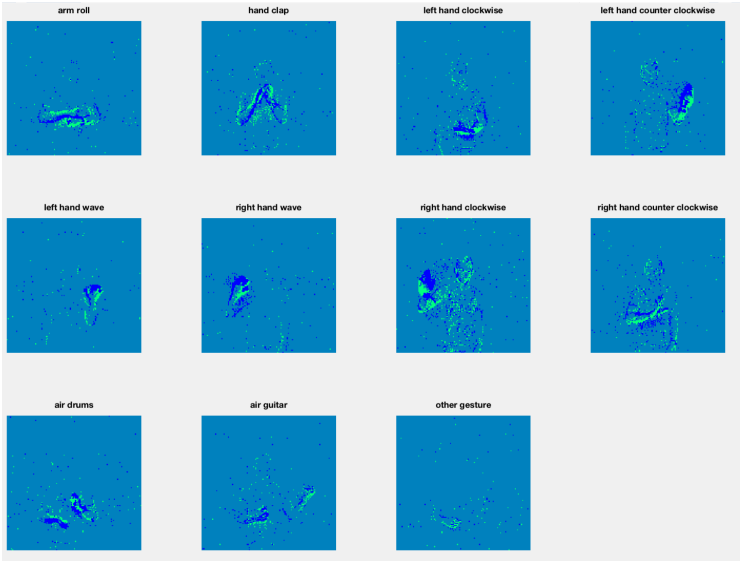


Figure 2.10: Visualisation of all gestures captured in the DVS128 Gesture dataset[6].

CIFAR10-DVS

CIFAR10-DVS[7] is an neuromorphic dataset for object classification. 10,000 frame-based images that come from the original CIFAR-10 dataset are converted into 10,000 event streams with an

event-based sensor, whose resolution is 128x128 pixels. The dataset has an intermediate difficulty with 10 different classes. A visualisation of these classes can be found in Fig. 2.11.

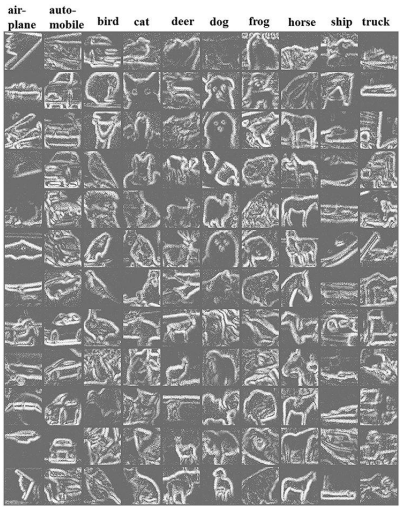


Figure 2.11: Visualisation of all objects captured in the CIFAR10-DVS dataset[7].

Event-Camera Dataset

The Event-Camera Dataset and Simulator[8] were created for the purpose For this purpose of motivating research on new algorithms for high-speed and high-dynamic-range robotics and computer-vision. It is a collection of datasets captured with a DAVIS in a variety of synthetic and real environments. In addition to global-shutter intensity images and asynchronous events, inertial measurements and ground-truth camera poses from a motion-capture system are provided. The latter allows comparing the pose accuracy of ego-motion estimation algorithms quantitatively. A visualisation of the frame-camera video alongside its event-camera counterpart can be seen in Fig. 2.12.

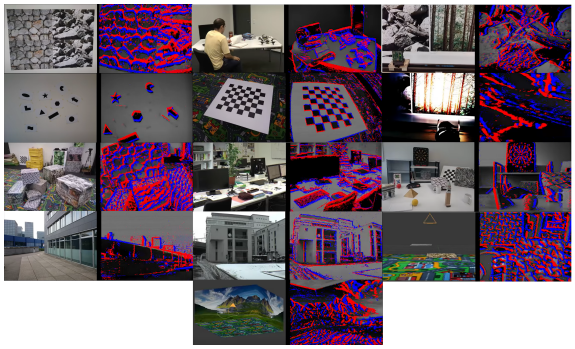


Figure 2.12: Visualisation of all videos and events captured in the Event-Camera dataset[8].

Heridelberg Spiking Datasets

The Spiking Heidelberg datasets for spiking neural networks[27] are useful for realising that spiking data is useful for so many more applications than computer vision. This dataset is split into two; The Spiking Heidelberg Digits (SHD) dataset and the Spiking Speech Command (SSC) dataset. Both of these datasets are audio-based classification datasets for which input spikes and output labels are provided.

2.7.2 Non-neuromorphic Datasets

Other data-sets such as fashion-MNIST could also be converted to spiking times by treating image intensities as input currents to model neurons, so that higher intensity pixels would lead to earlier

spikes, and lower intensity to later spikes, as was done by Nicolas Perez-Nieves *et al.*[12]. This avoids having to own an event-based camera to create data (as was done with the NMNIST dataset mentioned in Section 2.7.1), which is useful since the cameras are expensive and difficult to get a hold of in general.

Chapter 3

Analysis and Design

This chapter outlines overall design for the project as well as the analysis that formed the basis of each decision. The design of the project is mostly guided by the requirements set out in Chapter 1. Building on the work described in Chapter 2, novel pipelines are designed to advance the field of neuromorphic processing. The choice of programming language and other software/hardware tools, the processing of each dataset, and finally the structure of various classification pipelines are all set out before the implementation part of the report. As well as this a new frame-integration method is proposed, as well as state-of-the-art classification networks. The final part of this chapter also creates a framework for the comparison of each network, which can be found in Chapter 5.

3.1 Hardware and Software

3.1.1 Programming Languages

When choosing a programming language for the project there were a few choices that are most often chosen by developers; Python[28], R[29], and C++[30]. Python is the most popular choice due to the ease with which algorithms can be developed using it. Python features an extensive list of libraries and debugging functions that are invaluable when creating machine learning algorithms in particular, making the language of choice for this project. It is also important to note, however, the benefits of the other language options. C++ often results in programs with impressive performance due to the ability it grants to make low-level processes more efficient. Unfortunately this fine-level control also opens up programmers to more a demanding and time-intensive programming experience, with much more code writing and debugging to be done manually. R would also be a great choice for machine learning, and shares many similarities with Python, being open-source and having a huge community of developers constantly building libraries and tools. It has a different approach to machine learning, with a a more statistical analysis emphasis. Therefore Python remains the best choice for a more general approach for data processing.

3.1.2 Machine Learning Frameworks and Software

PyTorch

Pytorch[31] is a relatively new framework for machine learning, and provides a developer friendly way to write machine learning code. It has a more 'pythonic' approach to code abstraction than its competition in the space, and the *torch.nn.module* gives access to clear, reusable module definitions in an Object-Oriented Programming (OOP) manner. It also allows for simple data parallelism, so that batch processing can easily be split over different sets of hardware. It also has an intuitive debugging experience since it can use standard debugging tools such as PyCharm and pdb.

Tensorflow

Tensorflow[32] is the older and more widely adopted machine learning library. It provides a more robust set of functionality with clear documentation. In terms of deployment it is the clear favourite as it allows models to be deployed on specialised servers and even on mobile. When visualising data software such as TensorBoard are ideal as it includes functionality to display model graphs,

variables, histograms and much more. As well as this, Keras[33] is a framework developed by Google, and uses a primarily Tensorflow based back-end. It provides an easy to use API for fast prototyping and high levels of abstraction. It is used commercially by a plethora of companies and has a vast and highly developed research community. For these reasons Keras using a Tensorflow back-end were chosen for this project.

3.1.3 Other Software

SpikingJelly

SpikingJelly[18] is an open-source deep learning framework for Spiking Neural Network (SNN) based on PyTorch. It allows for the processing of many often-used datasets in the neuromorphic community (Some of which are described in Section 2.7), as well as a simple set of classes for building SNNs or converting ANNs to SNNs. The library, which was mainly co-developed by the Multimedia Learning Group, the Institute of Digital Media (NELVT), the Peking University and Peng Cheng Laboratory, can be installed directly via the *pip* command. Using it, data can be loaded as event streams, as well as integrated frames (described in Section 2.3.2) of varying frame lengths or frame-rates. As well as this the package features clear documentation and tutorials to begin analysing neuromorphic data.

NengoDL

NengoDL[34] is a software framework designed to combine the strengths of neuromorphic modelling and deep learning. It is a useful tool for constructing biologically inspired spiking neuron models, and combining them to create fully spiking neural networks. As well as this, these networks are intermixed with efficiently simulated deep learning concepts such as convolutional neural networks. This unified framework therefore allows us to train SNNs in the same way as we would for other ANN models with an easy to use interface. As well as this converters exist in order to convert ANNs to SNNs with relative ease.

3.1.4 Cloud Environments

Since Python is the language of choice for the project, Python Notebooks are a good choice for code segmentation and presentation. They allow for python code to be written in executable cells, so that their output, as well as other text, can be presented in a full document. This makes the code easy to understand for others, and good for development as well. Python notebooks can be run locally on a web server, as well as online on a cloud service. Many machine learning frameworks and algorithms make use of hardware acceleration using GPUs or TPUs. This allows for running code much faster on more powerful machines with this specialised hardware in them. This calibre of hardware was not readily available during the course of the project, so cloud services provided by the likes of Google and Amazon Web Services were used as a good alternative. They allow for the renting of GPUs etc. from their own servers, so that code can be run on them via their respective web services.

The two main web services available at this time are Google Colaboratory[35] and AWS Sagemaker Studio Lab[36]. In terms of hardware, both services offer access to powerful GPUs, though sagemaker offers the more powerful T4 GPU at the free tier. This benefit, however, is not entirely relevant as a pro subscription would be necessary with either service to make use high-RAM runtimes. Due to Colaboratory's better share-ability and wider adoption, it was chosen as the service for this project.

3.2 Datasets

The datasets used for the evaluation of the networks in this project were NMNIST (Section 2.7.1), DVS128 Gesture (Section 2.7.1) and CIFAR10-DVS (Section 2.7.1), since these are the most suited for training on classification tasks. As well as this their recording conditions and topics are varied, meaning an evaluation of the system can be more robust and reliable. The Event Camera Dataset for SLAM (Section 2.7.1) is also extensive and includes side by side frames from a traditional camera, making it ideal for testing the intensity reconstruction algorithm used in the two-phase pipeline described later in this chapter.

3.2.1 Data Pre-processing

Before processing data, z-score standardisation needs to take place. Equation (3.1) shows the equation dictating the process. In practical terms the standard deviation of x can be substituted by the max value of x. The reason this needs to occur is to 'center' the data, meaning the inputs to the network can be guaranteed to be of a certain scale. In the training process the weights and biases (as described in Eq. (2.4)) are applied to the initial inputs, which are then back-propagated based on the loss gradient. The standardisation process ensures that these gradients don't spike out of proportion, meaning the same global learning rate can be used for the network. As well as this parameters are often shared across deep learning networks, and if inputs weren't of a similar scale for every part of an image, for example, then this sharing of parameters wouldn't work since some areas would have a much larger weight than others. It is also important to note that the standard deviation and mean statistics used in this standardisation is always from the training dataset, since it is important to never use values from the testing set since that would invalidate it as a true indication of the network's performance.

$$x_{scaled} = \frac{x - mean_x}{std_x} \tag{3.1}$$

3.3 Two-phase Intensity Reconstruction Models

In order to make use of existing highly-tested and documented computer vision techniques, an intensity reconstruction of event streams was necessary. For this reason a two-phase network (shown in Fig. 3.1) was devised, wherein events are reconstructed into intensity videos before being passed into various classification networks. Each event stream was passed into the E2VID reconstruction network[4], allowing for each classifier that the data is fed into to analyse more complex visual patterns than were available from the frame-integrated event streams. The rationale behind this is that commonly used neural networks for video analysis are built to detect features from intensity frames rather than event streams. Therefore in order to make use of these specialisations, each event stream needs to be converted into a more common input form.

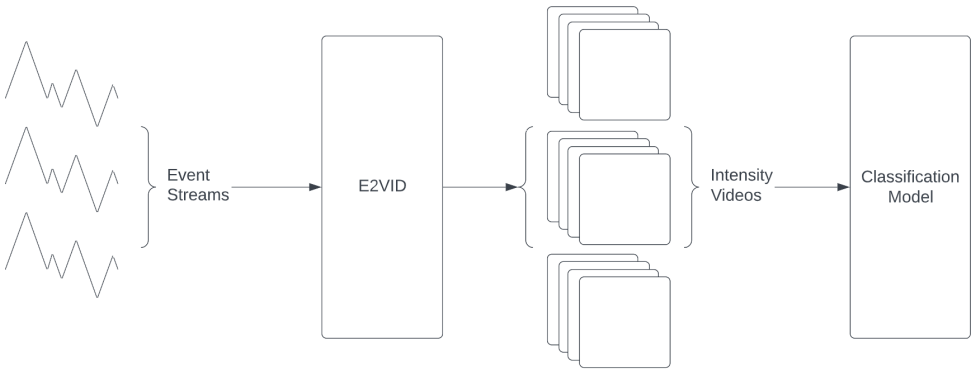


Figure 3.1: An illustration of the two-phase classification pipeline.

3.4 End-to-end Event Classification Models

For classifying frames directly, a frame integration method was utilised as described in Section 2.3.2. The method described involves having two channels per created frame. Each channel stores whether a positive or negative event was fired at any one pixel during a given time-slice. There are two variations of time-slice generation, which will be referred henceforth as 'synchronous' and 'asynchronous' frame-integration. Synchronous frame-integration is named due to the fact that each frame stores event information of a fixed time-frame. For example, each frame could store information for a 10ms time-slice. This means the generated frames are more akin to the outputs of frame-based cameras that have a fixed frame-rate. Asynchronous frame-integration, on the other hand, involves collecting a set number of events for each frame. The benefit of this method is that

the amount of information per frame is constant, irrespective of when more or less motion occurs in the video. Another asynchronous approach is to simply divide the events into a set number of segments. Then each segment can be integrated individually. These methods mean that certain frames may encode information over a longer time-span than other frames.

A slight adaptation of the classical method was also devised, in which rather than having binary information of on/off events per pixel of the frame, we could instead have a single channel value for each pixel with the sum of all event polarities for that location. This way the magnitude (or number) of events for each pixel over the time segment can also be captured. A diagram showing the basics of the custom integration process is shown in Fig. 3.2.

For classical frame integration the synchronous frame integration method was used, whereas for custom integration the second asynchronous method was used. Both these methods allow for the number of frames per sample to be constant, making it easier to input into neural networks.

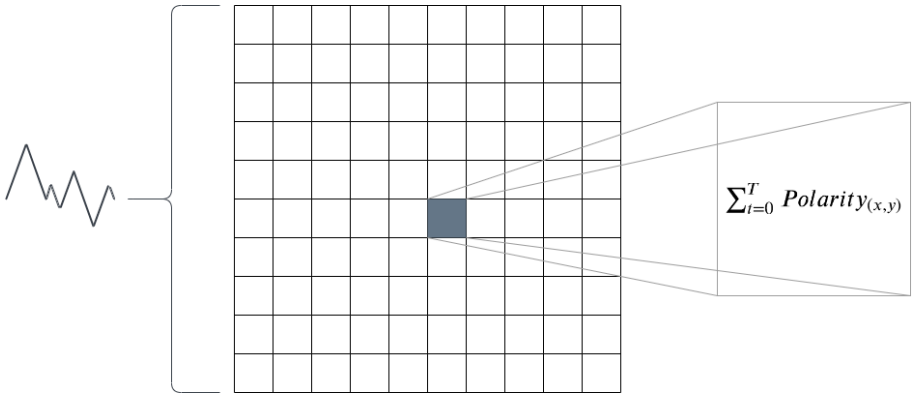


Figure 3.2: An illustration of the frame integration process with custom integrating method.

3.5 Classification Networks

Given below are the various networks used to classify either integrated frames or image reconstructions. Each network has its benefits and drawbacks, but there are a few commonalities between them.

During the training process of each of the networks a callback function was used called `EarlyStopping`. This built-in Keras callback allows for the interrupting of network training when the validation accuracy of the network stagnates or falls. This should, in theory, prevent over-fitting to the training data. A patience value is set so that even if accuracy drops for only one epoch it is given the opportunity to rise again. This, of course, means that the training dataset needs to be split into training and validation, where the validation set is more unseen data to evaluate the performance of the networks every epoch. Finally, the most optimal weights are loaded back when the network finishes training.

As well as this, dropout (or recurrent dropout for LSTM and GRU cases) layers were used. These layers mean that during the training process certain neurons will be 'turned off' and therefore not part of the back-propagation process. This should allow the resulting network to be robust since there is no over-reliance on any one neuron of the network.

Finally, for the learning rates for each system, it is necessary to find the optimal value. If the rate is too high, the model will quickly back-propagate to the optimal value, however once it gets close to it the weights may oscillate. This is because only a small step may be needed to make it to the ideal values, but the model constantly overshoots it. On the other hand, if the learning rate is too low, then the learning process will be very slow, and reaching an optimal value may take many epochs. As well as this, for more complex functions the network may get stuck on a local minimum value for the loss. It is clear in this case that an adaptive learning rate would be the most ideal. One would like the rate to be high to start off with, and become smaller and smaller as the training process goes on. This can be done with a Keras callback function, however for the models in this project the Adam optimiser[37] was chosen instead. This optimiser implements a function to find the best value for the learning rate itself.

3.5.1 3-Dimensional Convolutional Neural Network

A 3D convolutional network, as described in Section 2.6.1, was implemented to work on the event data. A diagram showing the function of one of the 3D convolutional layers used in the final network can be seen in Fig. 3.3. As opposed to their two dimensional counterparts, 3D convolutions involve taking video streams as input and output, allowing the network to decode patterns in both the spacial and temporal dimensions. As well as this the network employs batch normalisation layers in order to accelerate training. The reason the training time is reduced is that there is some regularisation, meaning the data is rescaled so that it centres around 0 and has a standard deviation of 1. In essence this means the outputs of each intermediate layer look similar in distribution, so the same learning rate can be used for back-propagation throughout the network.

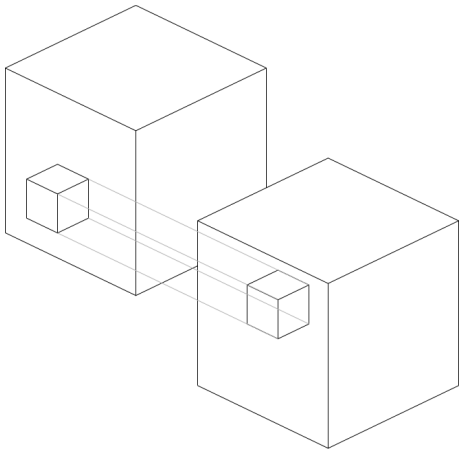


Figure 3.3: An illustration of a 3D convolutional layer.

3.5.2 Convolutional LSTM Network

A convolutional LSTM network as described in Section 2.6.4 was implemented to work on the event data. In this network the video that is fed in is passed through the network one frame at a time. The ConvLSTM layers apply 2D convolution on the frames to extract spacial patterns, and then these features are passed into LSTMs. The LSTMs allow for the network to remember previous frames and therefore analyse temporal features of the data as well.

3.5.3 Custom Convolutional LSTM Network

This network is an extension of the LSTM network in the previous section, adding more capacity for the model to learn spatio-temporal patterns. It involves altering the intermediate convolutional neural network that analyses each frame in the video time series. A diagram showing the pipeline of the overall network can be seen in Fig. 3.4. This additional capacity is helpful for deciphering more complex 'actions' in a video, such as in the case of gesture recognition.

3.5.4 Spiking Neural Network

With this network the whole system, from data generation to classification, could be fully neuromorphic. Using techniques as described in Section 2.2, traditional ANNs could easily be converted to SNNs[14] for classification. For the conversion process the `nengo_dl` package can be used (see Section 3.1.3). In order to train the network a differentiable approximation of the spiking neurons was used. This meant that the data could be trained using back-propagation as normal. During inference, however, the actual spiking neurons could be used[15]. Since the network is trained before the conversion from ANN to SNN, the performance of the SNN will not be comparable to the ANN due to approximation errors. These approximations can be improved by changing certain properties of the SNN and checking the performance of the inference.

Spiking neural networks work similarly to ANNs, however there are some key differences. One major difference is in the activation of the neurons. Whereas with non-spiking artificial neurons the output is a continuous function based on the instantaneous input, spiking neurons accumulate

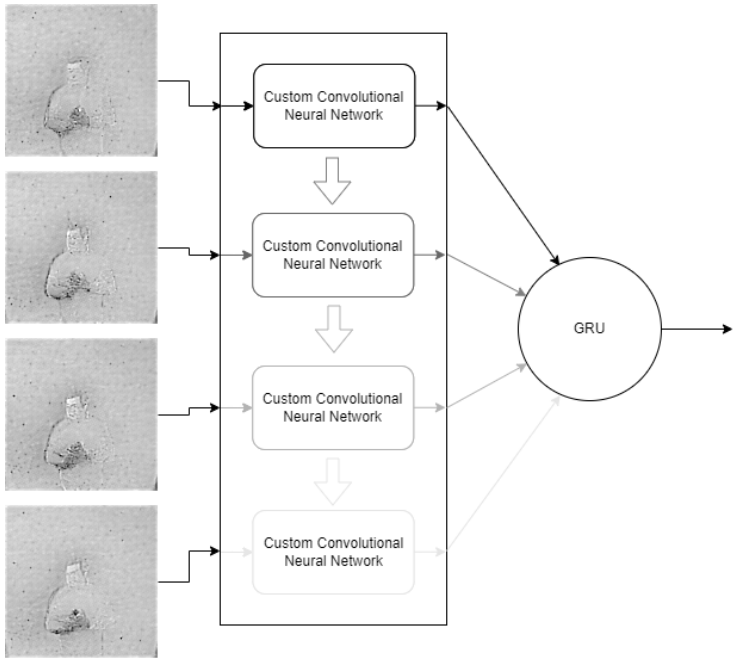


Figure 3.4: An illustration of the custom convolutional LSTM network.

inputs and build up to an eventual spike. This means that inputs to the neuron have to be sustained (repeated) for a certain amount of time to allow for this build-up of potential. For our network, since it is a 3D vector of frames over time, the input to the network needs to be a 4D vector. An illustration of the creation of the 3D vectors that were input into the ANN and the subsequent 4D network fed into the spiking equivalent are shown in Fig. 3.5.

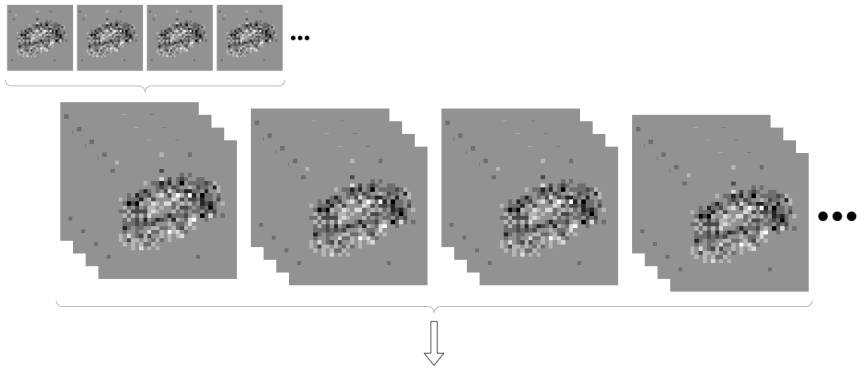


Figure 3.5: An illustration of how frames of a video can be repeated for input to a spiking neural network.

However, unlike convolutional network layers, recurrent layers such as LSTM and GRU do not have a direct conversion available to spiking networks. There is, however, a component known as Legendre Memory Unit (LMU) that are much better suited to the conversion process. Like LSTMs, LMUs have been shown to perform well in a variety of tasks with relatively few internal parameters[9]. TODO: Write more about the design of this network .

3.6 Evaluation Metrics

The evaluation plan is as given by the typical machine learning pipeline[38]. For a classification task, when we obtain the results from the test dataset (an example of which is shown in Table 3.1) we can calculate a variety of evaluation metrics that give various insights on our final model.

Labels	Predictions
1	1
1	2
3	8
9	9
6	9
⋮	⋮

Table 3.1: A table showing an example of results when inputting test data from NMNIST dataset[10] into the final model.

3.6.1 Confusion matrix

Confusion matrices act as a visualisation of a systems performance. It shows possible true labels as well as possible predicted labels on either side, and filled in are the number of results that fit in each segment. In Table 3.2 the confusion matrix for the NMNIST dataset is shown as an example. It should be noted that a similar confusion matrix should be created taking each class as positive, then each metric can be calculated by taking the averages (as shown in Section 3.6.6). For each of the cells the number of matching records are stored to calculate each of the evaluation metrics. The table includes True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN).

		Predicted Class			
		1	2	3	...
Actual Class	1	TP	FN	FN	...
	2	FP	TN	TN	...
	3	FP	TN	TN	...
	4	FP	TN	TN	...
	5	FP	TN	TN	...
	⋮	⋮	⋮	⋮	⋱

Table 3.2: a table showing one particular confusion matrix for NMNIST dataset[10] for class 1 as the positive class.

3.6.2 Accuracy

The accuracy of the system is the proportion of samples correctly classified.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Note: classification error can also be used and is defined as $1 - accuracy$.

3.6.3 Precision

Precision is the proportion of positively predicted samples identified correctly.

$$Precision = \frac{TP}{TP + FP}$$

It should be noted that a high precision may mean that there are many false positives.

3.6.4 Recall

Recall is the proportion of actual positives correctly classified.

$$Recall = \frac{TP}{TP + FN}$$

It should be noted that a high recall may mean a lot of positive samples may be missed.

3.6.5 F-measure/F-score

This is defined as the harmonic mean of precision and recall in order to get one number as an average measure of performance.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

3.6.6 Micro and Macro Averaging

Macro-averaging involves taking an average on the class level. Metrics are calculated for each class and then averaged at the end. Micro-averaging involves taking an average on the item level (i.e., taking the average of each of TP, FP, TN and FN to get the averages metrics).

Chapter 4

Implementation

This chapter presents the final implementation of the project. The full process for both the end-to-end event classification models, as well as the two-phase video reconstruction networks are given, alongside any data pre-processing that was required.

4.1 Data Preprocessing

As described in Section 3.2.1, data had to be centered to give a z-score. In practice the standard deviation was not used to normalise the values, and instead the following operation was undertaken: $\mathbf{x} = \frac{\mathbf{x} - mean_x}{max_x}$.

Due to the large size of each dataset, it was evident that the default Keras functions would not be ideal. This is often the case for real-world data meaning the it was far too large to store on system RAM. This is the case even with the high-RAM run-times of Google Coloboratory. This meant that data had to be read from files into the network on a per-batch basis, making the whole process less RAM intensive. Keras has a `Sequence` class that can be extended to create a custom data-generator class that could both read and pre-process the data directly. TODO: Maybe put code example here.

4.2 Two-phase Intensity Reconstruction Pipeline

An approach for event stream classification that can make use of existing networks in a literal sense is a two-phase video reconstruction pipeline. Here events are converted into a more familiar form so that patterns can more easily be found by convolutional networks.

4.2.1 Reconstruction Algorithms

E2VID

E2VID, as described in Section 2.4, is a state-of the art network based on UNET that reconstructs intensity videos from event data. Having gotten the output from the network (which on test data had an average Mean-Squared Error (MSE) of 0.05), it now needed to be processed slightly in order to work with the classification models. The main issue with the reconstruction was that the video were of varying lengths. In order to mitigate this, additional frames were added to videos shorter video reconstructions. These additional frames were added by repeating the video until the desired number of frames was reached. TODO: Maybe write more about this, and could add some images.

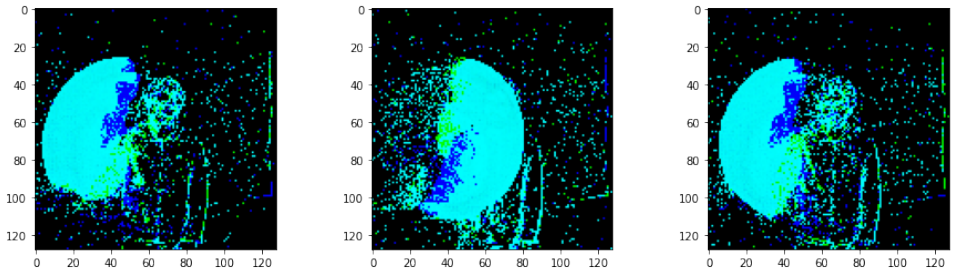
4.3 End-to-end Event Classification Pipeline

Frame Integration

In order to instead analyse neuromorphic data directly, it was pre-processed into a form that a NN can take as input. Initially, an implementation of the more formal method of processing events streams to get integrated frames given in Section 2.3.2 was implemented. This technique involved

creating two-channel images, with each channel holding binary indications of whether a positive or negative event was fired that given time-frame.

The integrated frames of a hand gesture in Fig. 4.1 shows the motion of an arm moving in a clockwise direction TODO: Show classic frame integration for NMNIST dataset . For each data sample, the event stream was split into 20 frames. This made the processing of the data easy for some of the networks described later in the chapter, since the frames could be packed into 3D tensors of equal size. It is interesting to note that the frames are not synchronous like they would be from a frame-based camera, and the amount of time represented by each frame is varying. As well as this, since the event streams are of varying length in the time dimension, some videos are reconstructed to a more granular scale than others. It is apparent that in this particular sample a relatively large amount of time is being compressed into every frame, resulting in a large amount of motion being visible. With more frames being created for each sample this problem would be alleviated and more features would be easily distinguishable. However if too many frames are taken, not only are processing times greatly increased, events may be sparse and not show any visible pattern in any one frame. A benefit of having this blurring event in the intensity frames is that the direction and degree of motion can be seen in the frames, and so the networks can also pick up these patterns in their classification process.



(a) (b) (c)

Figure 4.1: Three contiguous intensity frames **(a)**, **(b)** and **(c)**, created from the DVS128 Gesture dataset with a person moving their right hand clockwise.

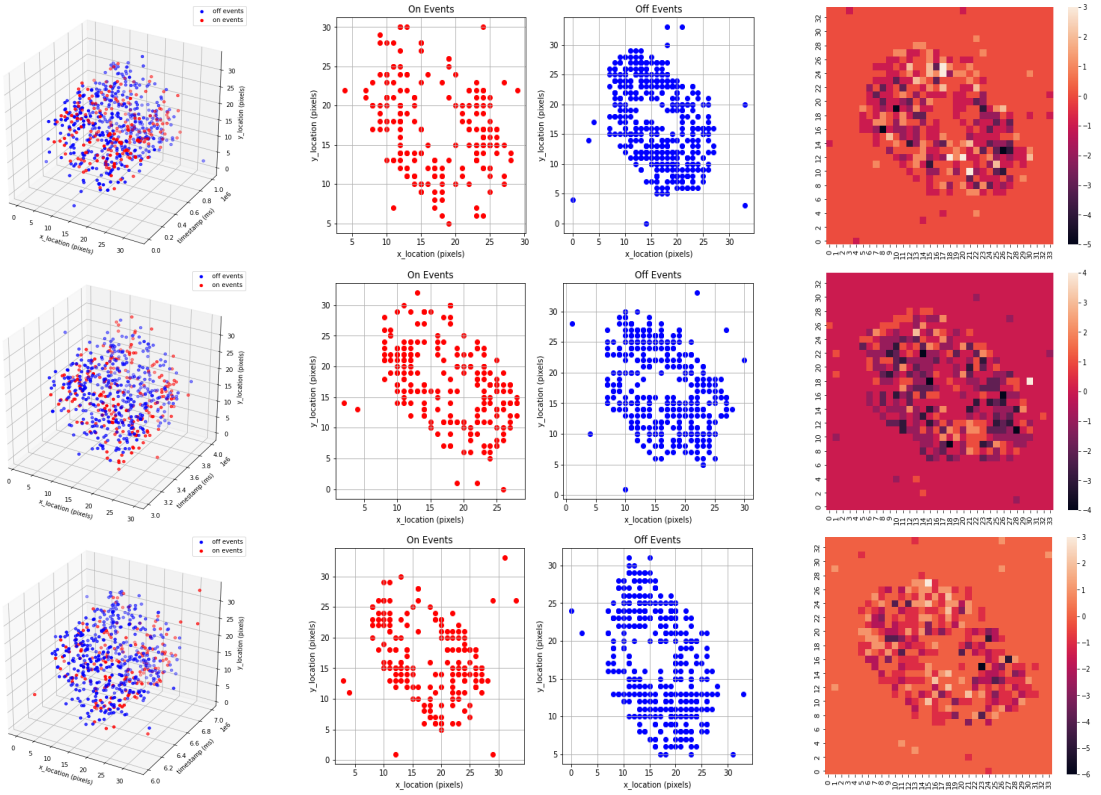
TODO: Add more photos of integrated frames with more and less frames

TODO: Add pseudocode for frame integration

Custom Frame-Integration

A novel integration technique was also implemented, as described in the Section 3.4. The method involved segmenting the events into groups based on their timestamp. Figure 4.2 shows a visualisation of intensity maps created from the NMNIST[10] dataset TODO: Show custom frame integration of DVS128 Dataset . The set of all events was split into eight segments, where each segment included events within a range of 1×10^6 ms (i.e., $0 \rightarrow 1$, $1 \rightarrow 2$, ..., $7 \rightarrow 8$). This way the data representation shifted to somewhat get back to a set of frames that mimicked the video output usually seen from everyday cameras. **(a)** shows the segmented events visualised in three dimensions (x_location, y_location and timestamp). In **(b)** these events were projected onto the two dimensional plane (of x_location and y_location), then the plots for on events and off events are shown separately. Finally in **(c)** an intensity map was created from the projected events. Each pixel in the intensity map grid was initialised to 0, and for every on event 1 was added to the cell, and for every off event 1 was subtracted from the cell. This way temporal information was retained to a greater degree than if each pixel simply had binary information of whether a event occurred or not in a two channel image (as was done by the spikingjelly package[18] in the previous section). It was clear that the resulting output greatly resembled the MNIST[26] sample recorded by the ATIS camera (As shown in Fig. 2.9 in Section 2.7).

TODO: Add stuff about possible nlp-like networks.



(a) (b) (c)

Figure 4.2: A visualisation of intensity maps created by segmenting events into bins of size 1×10^6 ms.

4.4 Classification Models

The models used to classify both the integrated frames and video reconstructions are given below. Please note that the input shapes given below are for the NMNIST dataset, however for the other datasets different input shapes would be required (e.g. {128, 128, 2} for the DVS128 Gesture dataset).

for each network the outputs of the latter hidden layers fed into a dense layer with 10 neurons (for NMNIST) to classify the correct class (0-9). Activation functions need to be present in the networks to prevent all the layers from becoming equivalent to a single one (linear regression model). In order to learn more complex patterns activation functions are a necessary aspect of creating an artificial neuron (See Eq. (2.4) in Section 2.2). The most commonly used activation function in deep networks (and image recognition in particular) is ReLU, so that was the natural choice for these networks as well. This activation function is also ideal for conversion into spiking networks (see Section 4.4.4). Finally, the output layers had a sigmoid activation function. This function compresses the output smoothly between the ranges of 0 and 1. this means each of the outputs from the neurons can be interpreted as a the probability of the input being any one of the output classes. This means we can simply take the highest probability as the predicted class from the network.

4.4.1 3D Convolutional Neural Network

The 3D convolutional neural network in Section 2.6.1 was altered to have the most appropriate parameters for each application. In order to choose the most appropriate parameters for the system, as well as the other systems implemented during the course of this project, multiple tests were run varying each of the possible hyper-parameters. The tests conducted were to determine;

- The number of hidden layers.

- The number of neurons per layer.
- The size of convolution kernels.
- The introduction of some fully connected layers after convolutional layers.

The network in each case was trained for multiple epochs. The performance of a typical network can be seen in Fig. 4.3, where the performance on the training set steadily improves as the network progresses through epochs. Accuracy is one of the metrics defined in Section 3.6, and the loss for the given network is called categorical cross-entropy loss. The formula used to calculate the loss is given by: $L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} log(\hat{y}_c^{(i)})$, where there are N samples and C classes. $y_c^{(i)}$ is 1 when the class is correctly predicted and 0 otherwise and $\hat{y}_c^{(i)}$ is the predicted probability of class c for data-point i. This is the same loss function that was used in the other networks in the report as well.

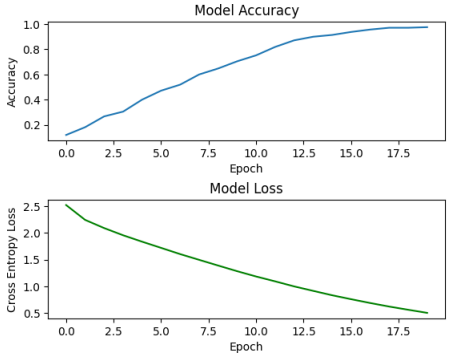


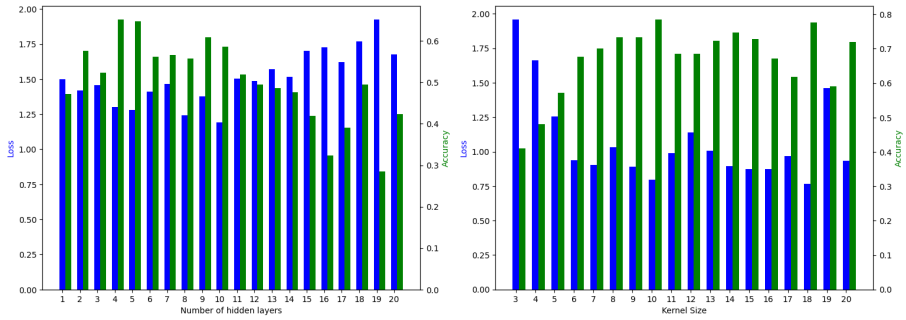
Figure 4.3: A figure showing classification accuracy and cross-entropy loss per epoch on training data for a typical network.

It was clear, however, that these results may be misleading since they only represent the efficiency of the system when classifying values within the training data-set. However, when looking at the performance on an unseen test data-set, it is obvious that some of the features learnt do not easily translate to general trends in unseen data. This is known as over-fitting, and can be avoided by reducing the capacity of the data-set so that it does not learn information specific to the training set, or by stopping the training process earlier. For this reason the `EarlyStopping` Keras callback function was utilised so that training would be stopped when over-fitting was detected.

Model capacity is directly correlated to the number of filters, as well as the number of layers, and as the model recognises more patterns in the training data, so it is important to get the optimum number for the system. The size of the kernels has an effect on the scale of the information picked up by the system. With smaller kernels more local patterns are detected, whereas with larger kernels more global effects can be seen. An example of the effects of changing such hyper-parameters can be seen in Fig. 4.4. It is clear from this that the performance of the network increases as more hidden layers are added, until past a certain point where over-fitting more occurs more often. As well as this there is an ideal kernel size for the scale of the data as well. As for the dense layers at the end of the network, it was found that better results were achieved as a result of them since global patterns could be further identified after the data had been processed by convolution layers.

Further testing was done with increasing kernel sizes, and with pooling layers added to the network.

An overview of the layers present in the final network, together with the shape of their outputs and number of trainable parameters can be seen in listing A.1. It features the repeating pattern of 3D convolution layers and max-pooling. The number of filters in each layer keeps increasing (from 32 to 64 and 128). The function of 3D filters is to capture patterns in the data, and as you move forward through the network these patterns get more complex. For example, if dots and lines are captured in the first layer, shapes such as triangles and squares may be captured in the second one (in the 2D case). Since the patterns are more complex, the number of possible combinations also increases. This is the reason more filters are required for later layers. The max-pooling layers scale down the image, this also means that more large scale patterns can be identified from smaller sections of the frame.



(a) (b)

Figure 4.4: Graphs showing the effect on training loss with varying hyper-parameters.

4.4.2 Convolutional LTSM

An overview of the layers present in the network, together with the shape of their outputs and number of trainable parameters can be seen in listing A.2. The thought behind the structure of this network is similar to the one when designing the 3D convolutional network. The number of filters increases as you go further through the network. The difference this time is that 2D convolution is carried out on each contiguous frame of the video input before being passed through LTSM networks. The way this is implemented is with the convLTSM network described in Section 2.6.4.

4.4.3 Custom Convolutional LSTM

An implementation of a custom convolutional LSTM can be seen in listing A.3 and the implementation of the custom 2D convolutional network applied to each frame of the video can be seen in listing A.4. This network is the natural progression from the previous convLSTM network since it applies a more complex 2D convolution network to each frame. This extra capacity is evident in the number of trainable parameters in the new network when compared to the one in listing A.2.

4.4.4 Spiking Neural Network via Conversion

In order to create spiking neural networks, the conversion process from Section 3.5.4 was used. First a 3D convolutional ANN was constructed and trained with traditional back-propagation. Once this training was complete, the network was converted to a spiking neural network. For this the activation of the neurons in the network were changed from tensorflow's `nn.relu` to nengo's `SpikingRectifiedLinear`. The specifics for the underlying conversion can be seen in the work done by Bodo Rueckauer *et al*[14]. The neural activity of this network can be seen in Fig. 4.5. The input to this network was an integrated frame video repeated multiple times to give a 4D tensor (as described in Section 3.5.4). The plots are shown over time, and since our network doesn't have any temporal elements (i.e. spiking neurons), the neural activity is constant for the whole duration. Since the neuron activations are continuous they each have a constant rate of neural activity. For this exploratory work the network was trained for 15 epochs, giving an approximate accuracy of 95.62%.

Then the network was converted to a spiking neural network by changing the Rectified Linear (ReLU) layers to Spiking ReLU layers. As previously explained, each 3 dimensional vector needed to be sent to the network multiple times to allow for spikes to accumulate in the neural network. It was found that to achieve a good number of spikes in the network the frames had to be repeated 30 times. A graph showing the activity of neurons in the first convolutional layer of the network with high and low number of repetitions can be seen in Fig. 4.7. Graphs showing the performance of the converted network can be seen in Fig. 4.6. Note that for this system since data it being passed into it consecutively, the only the final output from the system is sampled for predictions. The accuracy of the converted network fell to approximately 10%.

Once the conversion from ANN to SNN was complete, the performance of the network was incomparable to before the process. The reason for this (as outlined in Section 2.2) is the poor approximations that take place in the conversion process. In order to mitigate this, some network alterations were made to improve the approximations.

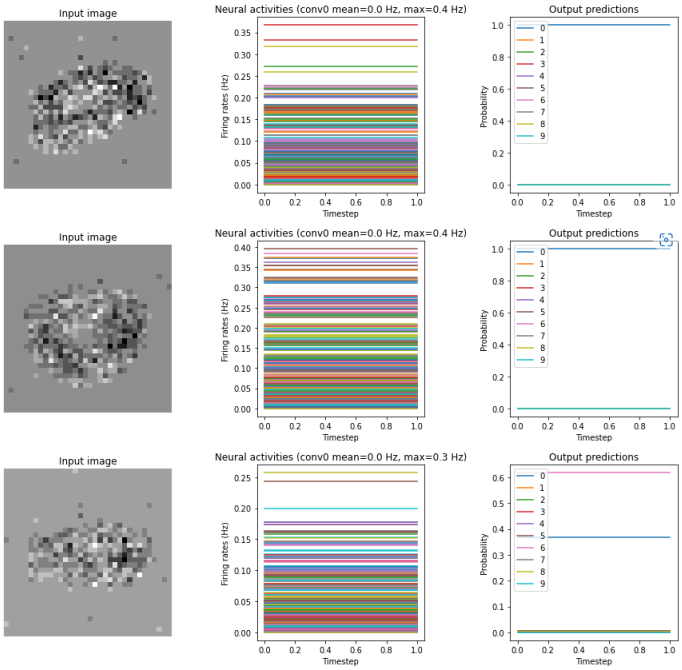


Figure 4.5: Graphs showing neural activity of ANN before conversion to SNN.

Synaptic Smoothing

The plots for the spiking neural network show that since the neuron function is now no-longer continuous, discrete spikes occur to propagate information throughout the network. This means that the output of the network is very noisy. Since there are so many neurons spiking at any one given time, there is no guarantee that when you sample the output at the final time-step that the correct neuron will be firing (even if it has a relatively higher firing rate). For this reason we need to apply a smoothing function to the firing neurons. Essentially the `synapse` parameter in nengo applies a moving average filter to the network output so that the predictions are more stable. The effect of applying this filter can be seen in Fig. 4.8. It was found that the performance of the system was best with smoothing=0.006.

Post-training Firing Rate Scaling

Neuron updates only occur when it outputs, or 'fires' a spike. This means that the system is much more likely to update when firing rates increase. Since the only output function that has been replaced is ReLU, which is a linear function, it is feasible to multiply the inputs of the neurons by a scale factor, then divide the output by the same number to scale. Larger inputs mean the neuron is more likely to spike, yet the linearity of the function is maintained by scaling down the output. The effect of applying this filter can be seen in Fig. 4.9. It is important to note, however, that if the firing rate were to theoretically be scaled infinitely, the neuron functions for be continuous, exactly replicating artificial neural networks. This may be good in terms of efficiency, but negates the point of using spiking neural networks in the first place. It was found that the best compromise for this system was to scale the firing rate by 150, which preserved the benefits of the spiking neural network and also improved accuracy back to the levels before the conversion.

It is also possible to change the effective firing rates of each neuron in the network, we could also add a custom loss function that incentivises a certain firing rate during the training processes as well. The benefit of this is that it can be applied to non-linear functions other than ReLU, but since there is no such function present in this project it need not be implemented.

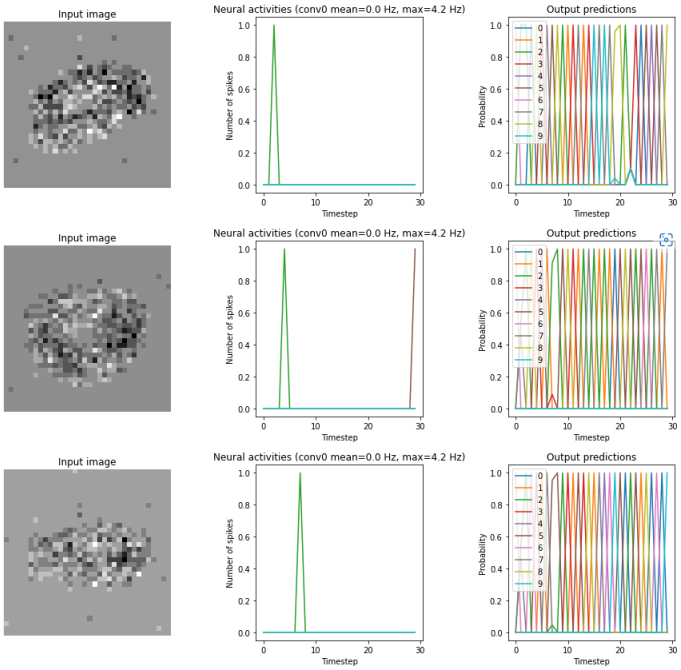
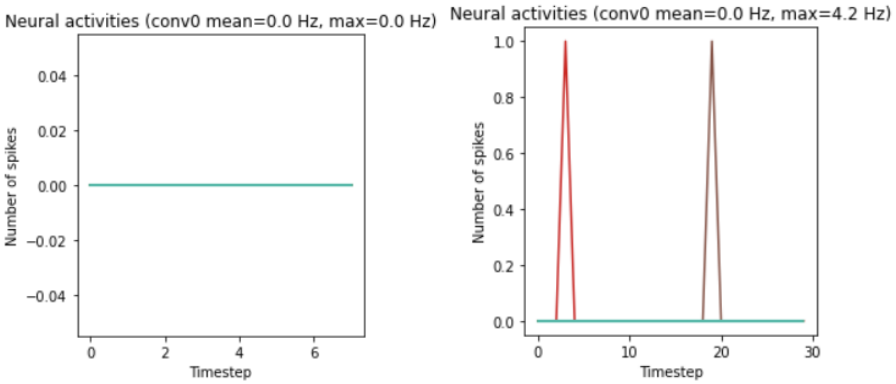


Figure 4.6: Graphs showing neural activity of ANN after conversion to SNN.



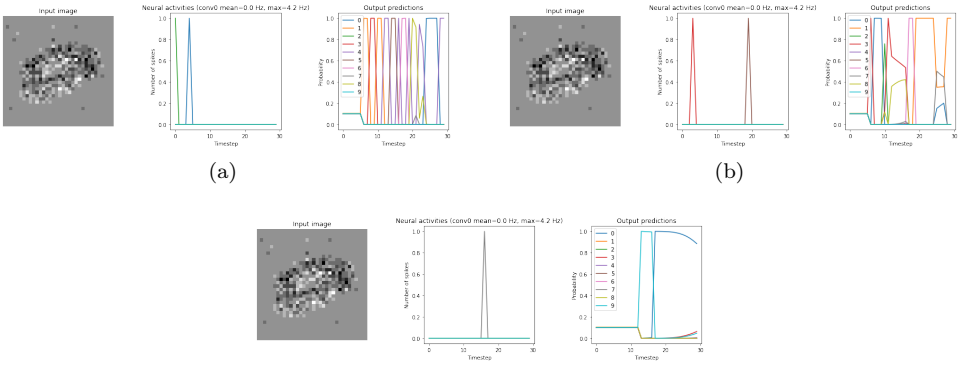
(a) (b)

Figure 4.7: Comparison of number of spikes with less **(a)** and more **(b)** repetitions.

4.4.5 Spiking Legendre Memory Unit Network

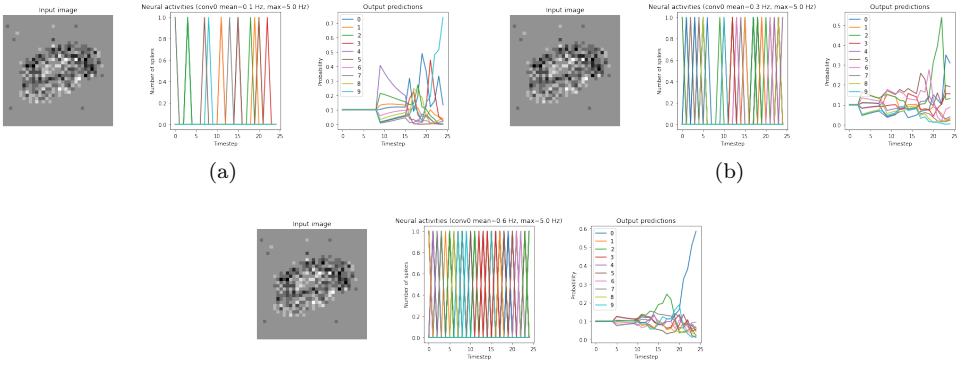
As mentioned in Section 3.5.4, there is no readily available conversion method for LSTMs or GRUs. LMUs, however, are ideal for converting into spiking nodes. An illustration of the dataflow through such a unit is shown in Fig. 4.10.

In the creation of of this unit in `nengo_dl`, a layer of neurons can be used for the linear and non-linear components. This way, each neuron in the layers can be converted to their spiking equivalent as with the 3D convolutional neural network. An adaptation of the code from the nengo documentation[39] was implemented. For this a `nengo.Ensemble` layer allowed for the use of spiking neurons, and synaptic smoothing could be achieved by adding filters to the outgoing signals (h_t).



(a) (b) (c)

Figure 4.8: Performance of spiking neural network with progressively more synaptic smoothing; **(a)** smoothing=0.002 **(b)** smoothing=0.005, and **(c)** smoothing=0.010.



(a) (b) (c)

Figure 4.9: Performance of spiking neural network with progressively more firing rate scaling; **(a)** scaling=20 **(b)** scaling=50, and **(c)** scaling=100.

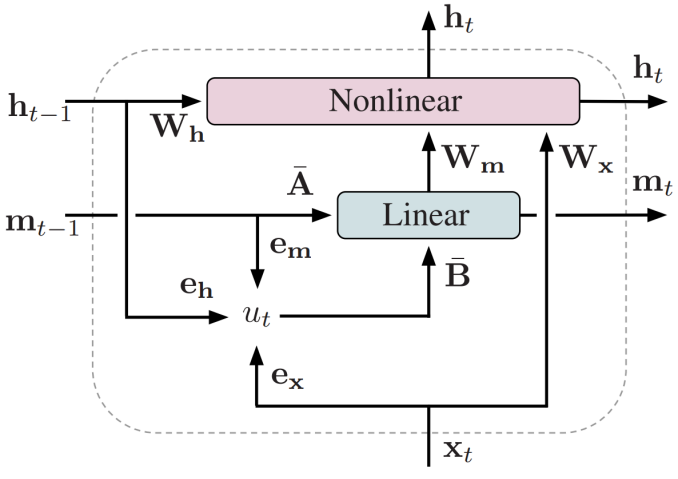


Figure 4.10: An illustration of the computational dataflows through an LMU[9].

Chapter 5

Testing and Results

In order to evaluate each of the networks implemented in the previous chapter, various tests were conducted. The main forms of evaluation used are outlined in Section 3.6. These metrics allow for a comprehensive understanding of network performance. As well as this some explanation of the findings are also given.

5.1 Data Pre-processing

The data was converted to the z-score as described in Section 3.2.1, which made the network learn more stably and ensure any patterns found were more robust and easily applicable to unseen data. This effect was much more apparent on the reconstructed video sequences since the intensities tended to vary much more than the intensities from integrated frames. This is since the values from the integrated frames were naturally 'centred' around similar values.

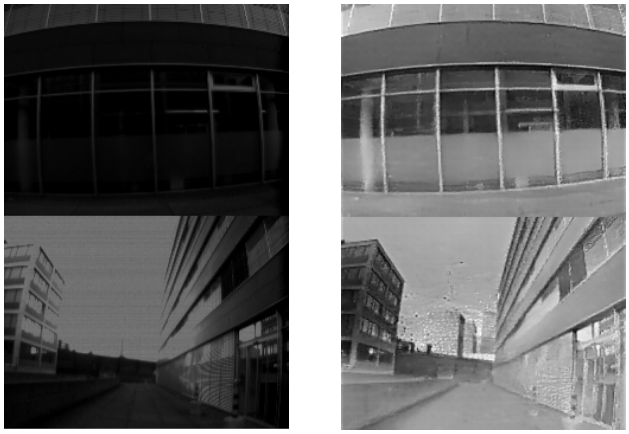
5.2 Two-phase Intensity Reconstruction Models

Most current classification networks are built to harness the features of video streams from frame-based cameras. To this end, networks were created to use these networks on intensity reconstructions from pre-built networks like E2VID (as described in Section 2.4). The results of such a network are given in this next section.

5.2.1 Intensity Reconstruction

The E2VID reconstruction network was used to recreate intensity videos from events. It was evident that the reconstructions created were robust and relatively to to life. The reconstructions were not effected by adverse lighting effects or fast motions, which can be verified in Fig. 5.1, which shows a reconstruction of a video of a runner in a sunny environment. It is evident that even in sunny conditions the event camera is able to capture high contrast details such as buildings in the background against the sunlight. Now, modern computer vision techniques could still be applied to event data, while still preserving the many benefits the event model presents. It was interesting to note the performance of the reconstruction model on inputs with few moving parts. Since events are only triggered when there is an intensity change on any given pixel on the sensor, only regions with motion in them showed up as events, and the nature of all the pixels with no motion was not easily inferable. This can clearly be seen in Fig. 5.2, where only parts of the scene were accurately reconstructed. This did not, however, pose much of a problem for tasks such as gesture recognition, since the motion is exactly what is being classified, however for other tasks such as object recognition it had to be ensured that there was some sort of motion of either the object or the camera for the reconstruction algorithm to be effective.

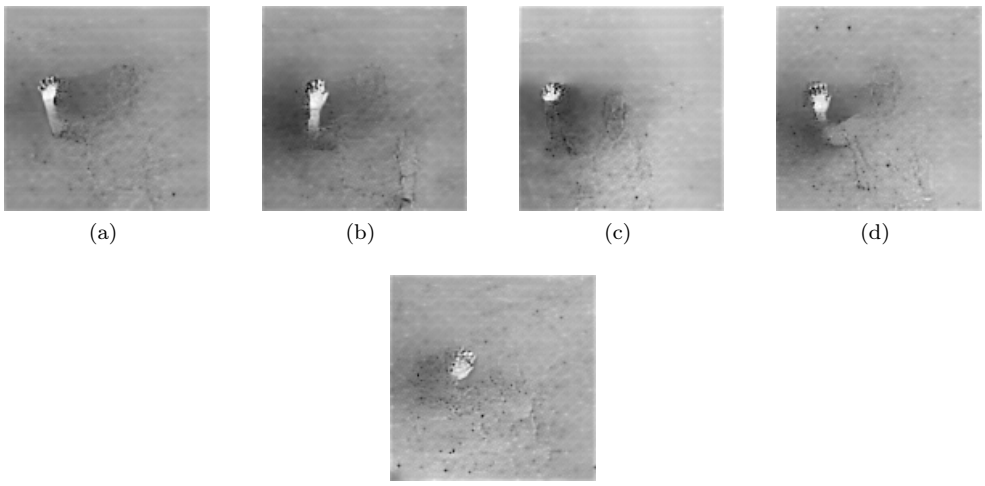
Figure 5.2 also shows that event-cameras do indeed allow for higher fidelity video capture in a wider range of lighting conditions that frame-based cameras (as explained in Section 2.1.1). In all lighting conditions the video reconstruction was largely the same, since the events triggered were very similar in all cases. The logarithmic characteristics of the event-sensor pixels are the reason for this, since the thresholds for the triggering of events is not static.Because the reconstructions are consistent across lighting conditions, this also means that the reconstruction is more reliable,



(a) (b)

Figure 5.1: Figure showing matching video **(a)** and reconstructed frames **(b)** from the event camera dataset[8].

and the classification algorithm works better in adverse conditions in general TODO: get figures and images to prove this . The E2VID reconstruction uses fixed-size event windows. This means that each input has n events, and therefore has an equal amount of information. This way the output data can be of a high fidelity even with large amounts of motion. For example event though the motion of the hand is very large in this video, the individual fingers and details are still clearly visible in every frame.



(a) (b) (c) (d)

(e)

Figure 5.2: A waving motion being reconstructed from events captures by DVS128 event camera under different lighting conditions.The lighting conditions are as follows; **(a)** fluorescent led, **(b)** fluorescent, **(c)** lab lighting, **(d)** led lighting and **(e)** natural lighting.

However, the quality of the reconstructions was not as impressive for datasets where the input is vastly different to the data E2VID was trained on (see Fig. 5.3). For example, with the NMNIST dataset[10], the input size was very small. This was a deliberate choice so that the format of NMNIST closely matches the format of MNIST, allowing for efficient training and testing of networks. However, the reconstruction of these samples was very rudimental, showing results similar to an edge map rather than a full reconstruction. This is one of the factors that made the two-phase pipeline less effective for this dataset as compared to the DVS128 Gesture reconstructions. This disparity could be eliminated if E2VID was re-trained on the specific data that was input, however for this project such datasets were not readily available, nor the time to train the network.

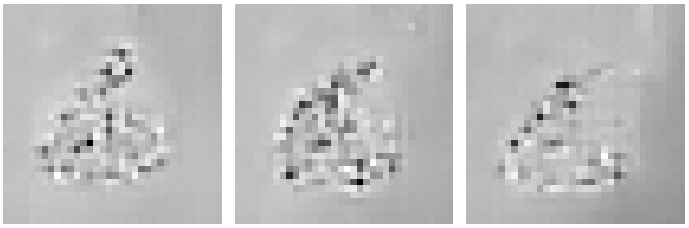


Figure 5.3: A figure showing three reconstructed frames from a recording of the MNIST character '6'.

Classification Results

The confusion matrices for the classification of the intensity reconstructed NMNIST dataset can be seen in Fig. 5.4. It is clear that the performance of the network is worse on more challenging characters such as 6, 8 and 9. This is understandable, considering the corresponding reconstructions of these classes is of noticeable poorer quality (see Fig. 5.3). Although the reconstructions are sometimes distinguishable by eye, the lines are often blurred. Due to the small scale of the dataset the resulting video stream is also of a small size, meaning less details can could be represented in the relatively low resolution images.

For the 3D convolutional neural network, the precision was often lower for these difficult classes. This means that these classes were over-represented in the predictions, when it should have been the other classes being detected (i.e., false positives). The purely convolutional network was able to identify patterns in the data due to its large number of trainable parameters, however this also meant that it over-trained on the features present in the more complex classes, often finding them in samples of the other classes.

Interestingly, with the LSTM networks it is instead the recall that was lower for these challenging classes, meaning many of the samples of these classes were misclassified (false negatives). The cause for this may be that the had fewer trainable parameters, but the recurrent capabilities made up for this allowing for the network to learn temporal patterns more reliably.

Overall the f1 measure acts as a balanced measure between precision and recall, since the data is not biased to any one class. The f1 score is superior for the 3D convolutional network and custom convolutional network, which both have similar performance on the test dataset. This makes sense since the temporal patterns are less important in the case of number classification, since motion is not prevalent or interesting.

Table B.7, Table B.8, and Table B.9 show the performance evaluation of each of the networks on the intensity reconstructed NMNIST dataset in more detail.

The confusion matrices for the classification of the intensity reconstructed DVS128 Gesture dataset can be seen in Fig. 5.5. Here the findings were a little different than for the NMNIST dataset. Firstly, it is important to note that when attempting to reconstruct the videos and pass them into each of the networks, the amount of memory often became an issue. This in itself is a clear indication of a large difference between the event and frame representations. After conversion, the frame-based video was of a much larger size and density than the event streams it was created from. In other words the efficiency of encoding was much higher in the case of event streams than intensity videos. For this reason it was not even feasible to run the ConvLSTM network with this data since both the memory and time requirements were much too high.

Between the purely convolutional and LSTM network there were still many interesting conclusions to be drawn. Th performance disparity between the two was of a much larger scale than with the reconstructed NMNIST dataset. The reason for this is that the recurrent neural network was much better at recognising temporal patters as well as spatial ones in the data, which is much more important in the case of gesture recognition. For this reason the overall accuracy was much better with the LSTM network. For cases with very similar frames (such as clockwise and counter-clockwise arm rotations), the LSTM had far better precision as well.

Table B.13 and Table B.14 show the performance evaluation of each of the networks on the intensity reconstructed DVS128 Gesture dataset in more detail.

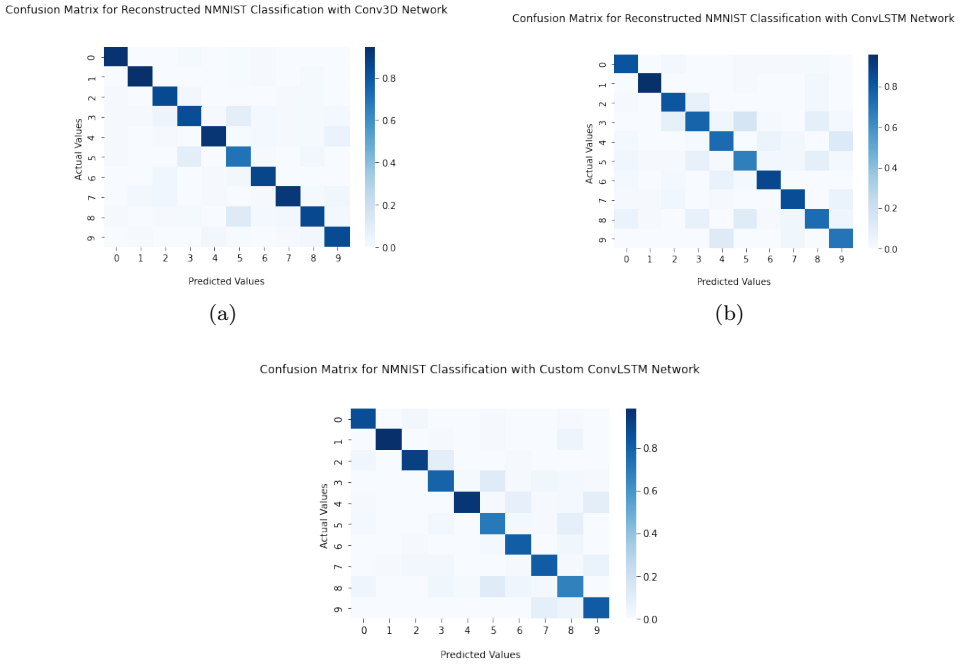


Figure 5.4: Confusion matrices for intensity reconstructed NMNIST classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

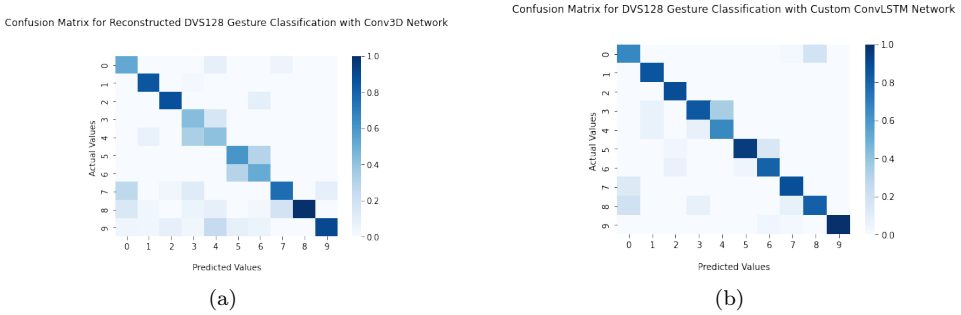


Figure 5.5: Confusion matrices for intensity reconstructed DVS128 Gesure classification with various networks; **(a)** conv3D, **(b)** custom convLSTM.

5.3 End-to-end Event Classification Models

This section shows the results obtained from the end-to-end event classification pipeline. The benefits of the event driven camera (as described in Section 2.1.1) were evident in the acquired results. As opposed to tradition frame-based cameras, high frequency data is not lost when processing events. There are spikes for every event at a much more granular scale in the temporal dimension in the event camera when compared to the frame camera, meaning fast movements were captured more reliably since events are captured at the μs scale, no longer restricted by the frame-rates of modern cameras (which often results in motion blur).

5.3.1 Frame Integration

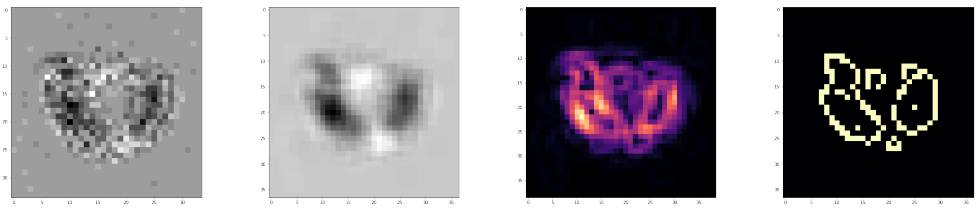
It is evident that modern computer vision techniques have been developed with frame-based cameras in mind, and so modern networks achieve good accuracy, and are able to find patterns well, on frame-based data. For this reason the common technique of integrating frames Section 2.3.2 results in frames, akin to the ones a regular camera generates. It does, however, still pose many benefits when compared to the frames from a regular camera. As mentioned previously information between

frames is still not lost or degraded since the events are still captured and visible in each frame. As well as this, the frames generated from events inherently focussed on the points of interest in the image, since these were the only ones in motion in the frame. Most common architectures (such as the one proposed by Raimundo F. Pinto *et al.* for static hand gesture recognition[40]) feature an intermediate layer to remove backgrounds and other noise from images to focus on the points of interest. With frame integration, this intermediate layer could be omitted, since the output was already similar to an edge map. It is conceivable, however, that in noisy environments with lots of motion this stage would still be necessary.

As previously mentioned, when using the frame integration method the result was very similar to an edge map, which is often the primary step of image analysis using convolutional neural networks to images in existing networks already. Figure 5.6 shows the effect of carrying out canny edge detection[41] on an integrated frame from the NMNIST dataset. The sample is taken from a recording of the number '0', and it can be seen that the original frame is in essence just a noisy edge map of the number. The steps taken for canny edge detection were as follows;

1. Smooth image by convolving with an averaging filter of the form: $\begin{bmatrix} \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} \\ \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} \\ \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} \\ \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} & \frac{1}{4^2} \end{bmatrix}$
2. Sobel edge detection was carried out in order to find the edges of the smoothed image. To do this the image was convolved with the following matrices: $S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ and $S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$. These matrices got the pixel gradients in the x and y directions, and taking their magnitudes ($\sqrt{S_x^2 + S_y^2}$) gives the edges map of the frame.
3. Hysteresis thresholding allowed for weaker edges to be counted as long as they are connected to stronger ones. Then non-maximum suppression was applied to get a single line for every edge (by finding the strongest point of every line in the direction of its gradient).

This refined edge map could have been part of the pre-processing of the data before being passed into the network, but it was found that this was not beneficial to network training. This was because some information is lost in the smoothing process, and any feature mapping was done efficiently during the training of convolutional networks anyway.



(a) (b) (c) (d)

Figure 5.6: Progression of canny edge detection on integrated frame of a sample from the NMNIST dataset with class '0'. The steps are as follows; **(a)** original integrated frame, **(b)** smoothed, **(c)** edge detection, and **(d)** Non-maximum suppression and hysteresis thresholding.

Classification Results

The confusion matrices for the classification of the frame-integrated NMNIST dataset can be seen in Fig. 5.7. In terms of precision of classification, all networks perform relatively well. However, the 3D convolutional network and the custom convolutional LSTM network were marginally more effective, achieving a higher accuracy and correctly classifying more challenging event streams. Where this is most apparent is when classifying numbers such as 3, since with the base convolutional LSTM network these were sometime misclassified as an 8 or 9.

The performance of each network on the frame integrated NMNIST was better overall than with the intensity reconstructed equivalent (as is evident by the superior f-score of the classification networks), which may be due to the fact that temporal information was not lost, as was the case in the reconstructed video stream. Due to the poor performance of the intensity reconstructions overall, it was outperformed in most, if not all cases.

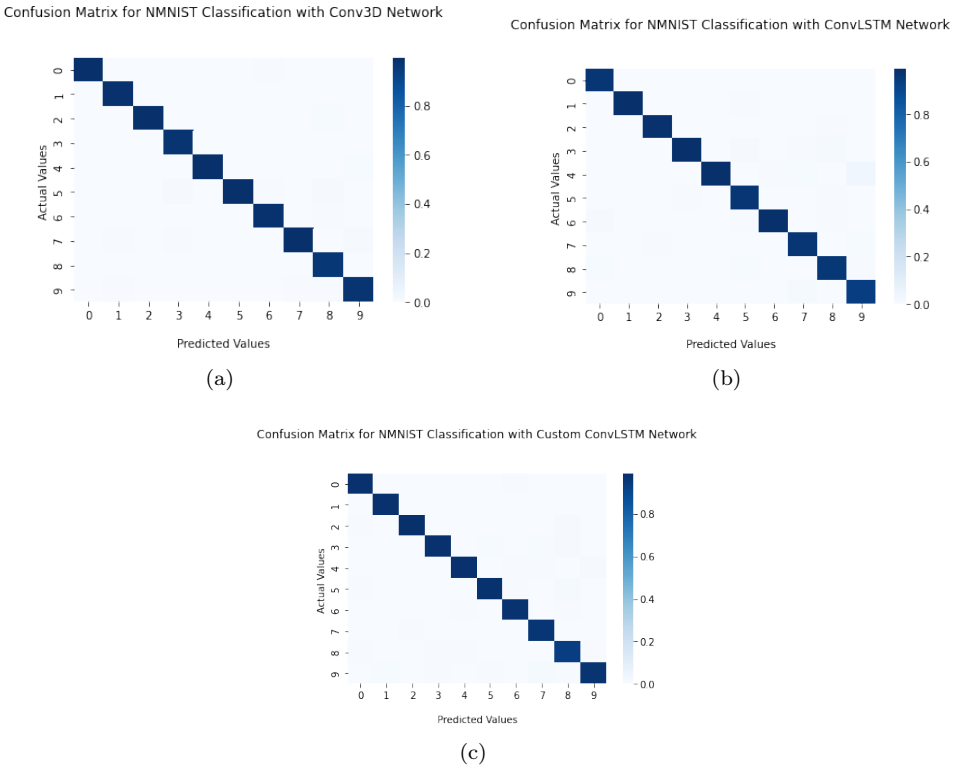


Figure 5.7: Confusion matrices for frame-integrated NMNIST classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

Table B.1, Table B.2, and Table B.3 show the performance evaluation of each of the networks on the frame-integrated NMNIST dataset in more detail. Once again the network has trouble classifying the difficult classes. We can see that the performance suffers more when compared to the networks analysing the integrated frames. Especially in the NMNIST dataset the edges are very significant for distinguishing between each class, which integrated frames excel at highlighting. The additional details added to the frames by E2VID seem to only hinder performance.

The confusion matrices for the classification of the frame-integrated DVS128 Gesture dataset can be seen in Fig. 5.8. The variance in performance between the networks is more apparent in this dataset than for the NMNIST dataset. The reason for this is that not only is object detection (of the human body) being undertaken by the system, but also action recognition across frames. It is evident that actions 4 and 5 (right arm clockwise and right arm counter-clockwise), as well as actions 6 and 7 (left arm clockwise and left arm counter-clockwise), were often misclassified as one another. The reason for this was that with length 20 frame integrated videos the fast rotational movement results in frames that look very similar in both directions TODO: Maybe try to get some pictures to prove this . This could be remedied by having more than 20 frames for each integrated video sequence. Other mistakes were more often made with the 3D convolutional network. For example 10 (air guitar) was quite often predicted for cases of classes 8 and 9 (air roll and air drums).

Table B.10, Table B.11, and Table B.12 show the performance evaluation of each of the networks on the frame-integrated DVS128 Gesture dataset in more detail.

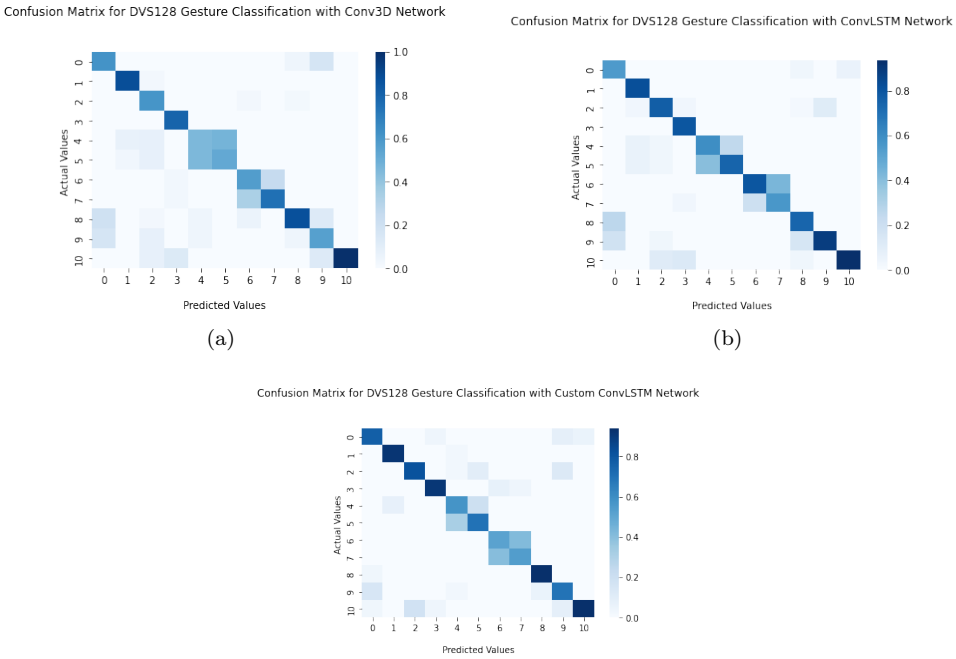


Figure 5.8: Confusion matrices for frame-integrated DVS128 Gesure classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

5.3.2 Custom Frame Integration

Some tests were undertaken using the custom frame integration technique on the DVS128 Gesture dataset. With this method, information can be stored to the same, or even smaller, temporal resolution in a lower channel image. Fig. 5.9 shows the left arm clockwise hand rotation gesture represented using the two techniques. It is clear that the encoding is more efficient, since only single-channelled greyscale images are required with the custom integration method, and more fine-grained temporal information is retained since the intensity of the pixels signifies the number of events occurring in any given pixel. For example, in **(b)** it is possible to visually know where the position of the arm is, as well as the movement it was going through without having to create too many individual integrated frames. We can see that when 20 integrated frames were created for the NMNIST dataset with the classical method, the motion is visible in one large blur, with no clear indication of the position of the arm.

TODO: Write some experiments for comparison of the two techniques.

Classification Results

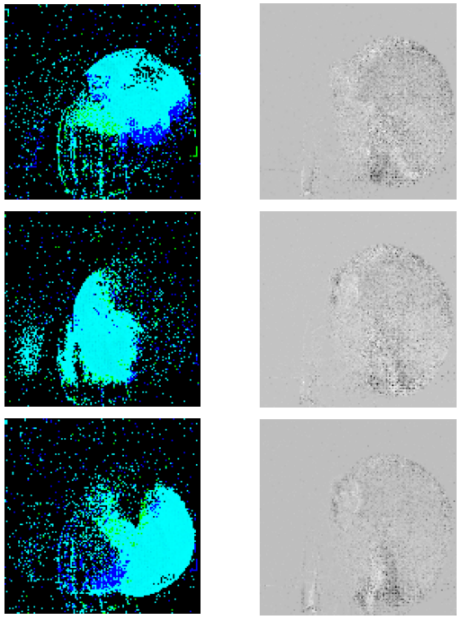
The confusion matrices for the classification of the frame-integrated NMNIST dataset with the custom frame integration method can be seen in Fig. 5.10. TODO: Write more about this and redo confusion matrices .

Table B.4, Table B.5, and Table B.6 show the performance evaluation of each of the networks on the custom frame-integrated NMNIST dataset in more detail.

TODO: Get result for custom frame-integrated DVS128 Gesture.

5.4 Comparison of Classification Networks

The full table of classification accuracies can be seen in Table 5.1. Here, the different strengths of the various networks becomes apparent in this comparison. For the simple object classification task on the NMNIST dataset, the higher capacity 3D convolutional network outperformed the others. However, in the more complex task of gesture recognition on the DVS128 Gesture dataset it struggled. The reason for this is that though it is excellent at detecting spacial patterns in each



(a) (b)

Figure 5.9: Frame integration of arm rotation gesture from DVS128 Gesture dataset with two different methods; **(a)** classic frame integration, and **(b)** custom frame integration.

Dataset		Conv3D	ConvLSTM	Custom ConvLSTM
NMNIST	Intensity Reconstructed	86.67%	79.60%	83.20%
	Frame Integrated	98.74%	97.64%	97.70%
	Custom Frame Integrated	98.22%	97.48%	97.88%
DVS128 Gesture	Intensity Reconstructed	63.26%	-	82.95%
	Frame Integrated	69.44%	71.53%	76.39%
	Custom Frame Integrated	69.32%	72.80%	80.14%

Table 5.1: A table showing classification accuracies of various models.

frame, it is less able to detect the temporal patterns prevalent in the gestures. For these cases the custom convLSTM network shone, outperforming the 3D convolution network by 20%.

The 3D convolutional network had the most trainable parameters, which afforded it better performance on the NMNIST dataset. The dataset's scale is small enough for the network to learn all possible features and get a good classification accuracy. For the more complex cases of gesture recognition, however, the performance of this network lagged behind the recurrent LSTM networks (especially the custom one). The reason for this is that in these cases temporal patterns are much more important, and LSTMs are ideal to learn such features. The 3D convolutional network could no longer memorise enough features to accurately classify all gestures, and even often over-fit leading to lower accuracies on the unseen testing data.

The training times for each of the networks on the datasets can be seen in Fig. 5.11. It is evident that the networks with a higher capacity had higher training times. The 3D convolution network and custom convLSTM networks in particular had a longer training times due to having many more parameters to optimise via back-propagation. It should be noted that for the networks in missing classification accuracies in Table 5.1 the system often encountered an Out Of Memory (OOM) error when attempting create a tensor of larger batch size. For these cases a smaller batch size was used, resulting in slightly higher training times, as well as possible skewed results. If even this was not sufficient, results were not taken for these cases.

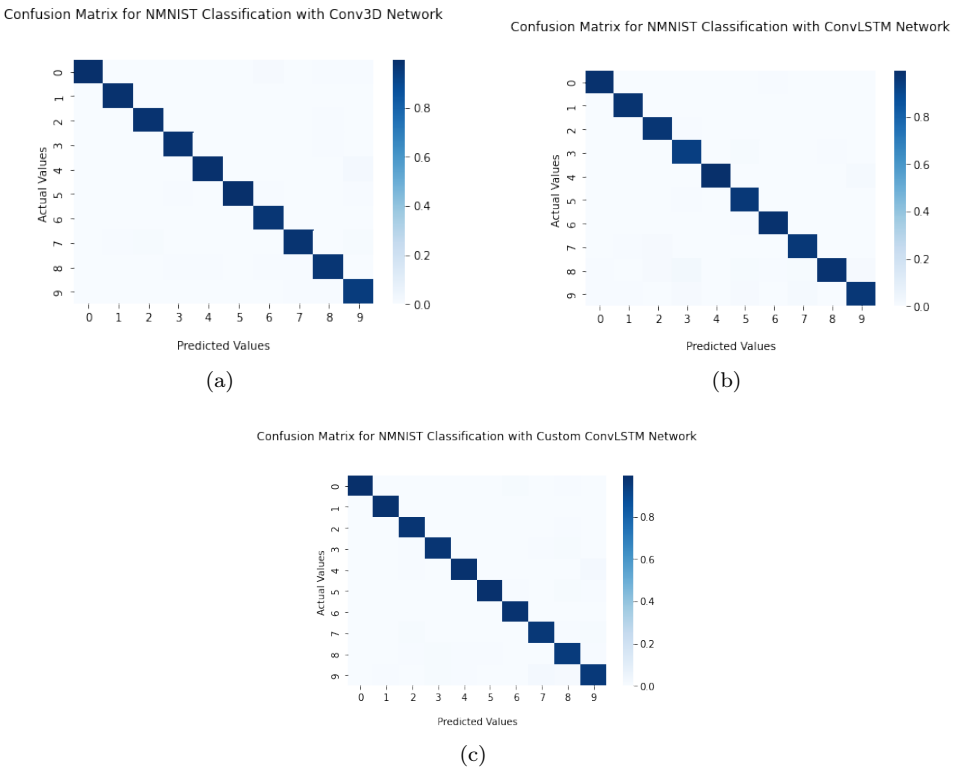


Figure 5.10: Confusion matrices for custom frame-integrated NMNIST classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

5.5 Comparison of Event Analysis Pipelines

Interestingly, there are some discrepancies in the performances of each pipeline when they are doing simple object detection and more complex tasks such as gesture recognition. Some comments about this discrepancy are given in this section.

When comparing the two-phase video reconstruction pipeline to the direct event classification one, the results show that for the NMNIST dataset, the direct event analysis leads to superior classification accuracies. The reason for this could be the loss of some high frequency data when creating intensity reconstructions with event data. The reason for this is that the created video is of a certain frame-rate, which is much lower than the temporal resolution of events captured by event cameras (which are on the μs level scale). There is, however, scope to still use the reconstruction pipeline without losing too much temporal information. For this the number of events per input to the E2VID network can be reduced so that the resulting video is of a higher frame-rate. However, this would require for the network to be retrained to achieve good results on the data with fewer data-points per frame.

Moreover, in the case of NMNIST, the reconstructions were not too dissimilar to the integrated frames. In both pipelines the input was similar to an edge map, and so the performance difference was less noticeable. This may be because of the small size of the dataset frames, and so for better reconstruction performance the E2VID network would have to be re-trained. Overall the performance of the two-phase pipeline was lower than working directly on the integrated frames since even though both inputs were like edge maps, the integrated frames retained far more temporal data.

For the DVS128 Gesture dataset, the performance difference between the two pipelines was far smaller. In this case the reconstructions were much more realistic and well-defined, allowing for the classification networks to more easily find patterns in the videos. Unlike in NMNIST, where the direct event classification pipeline outperformed the two-phase intensity reconstruction pipeline, for gesture recognition the opposite was true. The reconstructions were of particularly good quality for the DVS128 Gesture dataset, allowing for the classification networks to gather useful patterns. The reconstructions were also similar for all lighting conditions, meaning performance was consistently

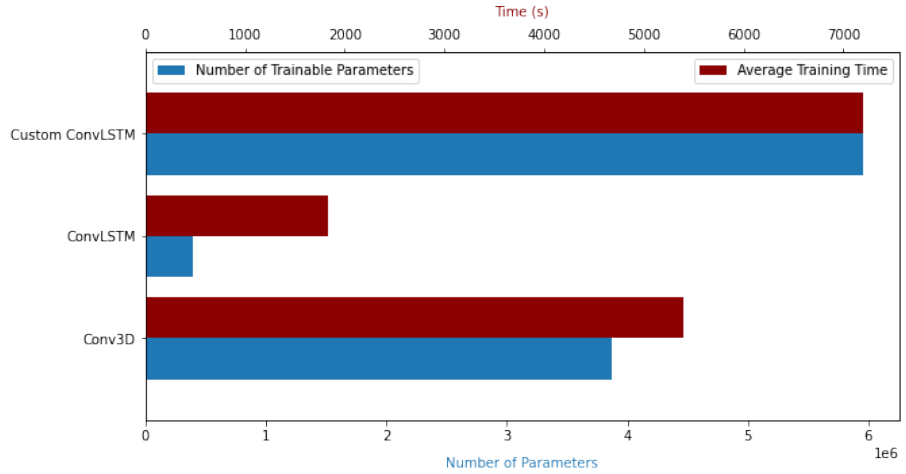


Figure 5.11: A figure showing the training times and number of trainable parameters of each network.

strong.

Overall, for more complex tasks such as gesture recognition, video reconstruction had better performance than with more simple classification problems. It is fair to assume this trend extrapolates to more complex classification tasks, since the intensity reconstructions add more detail to frames than are available in the events they process. As well as this, when tested on the Event Camera dataset, the reconstructions were much clearer visually than the respective video frames. As well as this the more complex details from the environment were also reconstructed, meaning the there is more information held in the reconstructed frames than in the events they were generated from. This could be incredibly beneficial for more complex and fine-grained classification tasks.

5.6 Evaluation Spiking Neural Network Conversion

5.6.1 Converted 3D Convolutional Network

As mentioned in Section 4.4.4, there is a compromise to be made when attempting to scale neuron firing rates to get a better accuracy. When firing rates are scaled past a certain level, the network becomes equivalent to a non-spiking network. This would mean the loss of the benefits of sparse encoding the the SNN. The effect of over-scaling firing rate can be seen in Fig. 5.12, where a very high firing rate results in a network very similar to a non-spiking neural network.

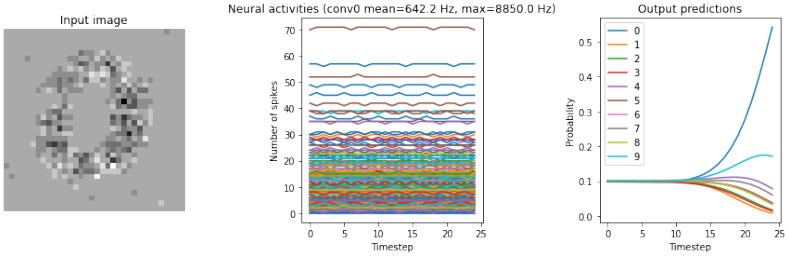
The relationship between firing rates and performance can be seen in Fig. 5.13, where the accuracy gets closer to the non-spiking network as the spiking rate is artificially inflated to infinity. It was found that for the conv3D network, comparable performance could be obtained while keeping the data-flow through each network quite sparse, since close to 100% relative performance could be reached with an average neuron spiking rate of less than 1Hz for both the NMNIST and DVS128 datasets. Especially for the DVS128 dataset, with very few average neuron spikes, the network achieved good relative performance.

5.6.2 Legendre Memory Unit Network

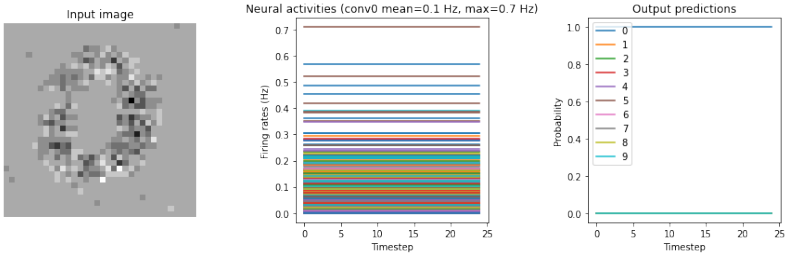
Network	NMNIST	DVS128 Gesture
Non-spiking LMU	90.2%	60%
Spiking LMU	60%	

Table 5.2: A table showing the performance of LMU networks on various datasets.

Table 5.2 shows the promise of LMUs as a recurrent module for spiking neural networks. The



(a)



(b)

Figure 5.12: Comparison of spiking neural network with firing rate over-scaled **(a)** and a the equivalent non-spiking neural network **(b)**.

performance before SNN conversion is similar to LSTM networks, and could feasibly improved with convolutional elements as was done with the convLSTM layer.

TODO: Write about difficulties with spiking LSTM and experiments with LMU network.

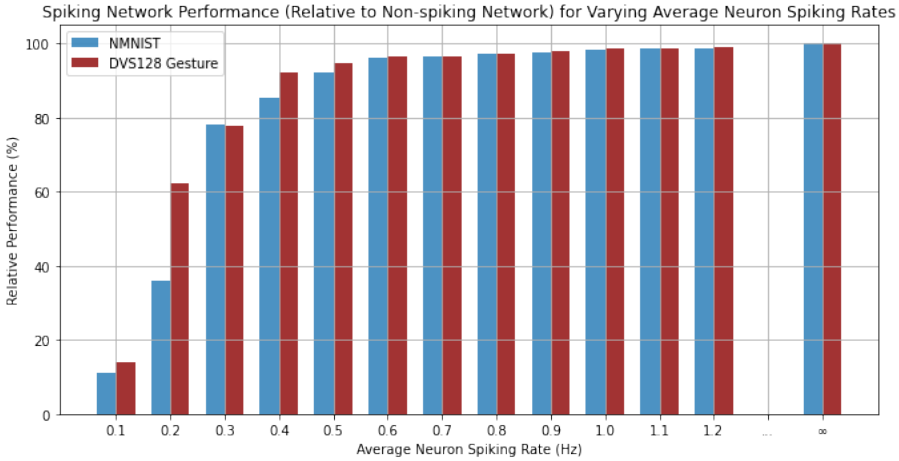


Figure 5.13: A figure showing the performance of the spiking 3D conv network relative to its non-spiking equivalent with varying spike rates.

Chapter 6

Conclusion and Further Work

To conclude, the testing done with the two classification pipelines show that there are significant benefits to working within the neuromorphic paradigm as opposed to with classic frame-based cameras. With the same networks, performance appears to be superior when analysing event streams rather than video frames. Secondly, the two-phase intensity reconstruction pipelines show promise, being able to harness the wealth of existing networks that exist for video classification. However, the performance with the same classification networks directly on the events rather than the video reconstructions tended to have better performance, perhaps due to the loss of high-frequency data and when creating the intensity maps. This is however open to interpretation, since the models used to classify the reconstructions are relatively simple. The reconstructions do show that the benefits of event-based cameras can be retained even when working with intensity videos, as the reconstruction networks produce high fidelity outputs even when recording videos with lots of motion or high dynamic range of light intensities. This means that the reconstructed videos are often more appropriate for classification that if a video was taken with a frame-based camera. TODO: Write about performance of spiking neural networks.

6.1 Further Work

Having reached the end of the project, it is important to note the limitations (such as computing power and available time) that need to be acknowledged. As such, there are some key areas in which there is scope for further research and development. These include;

1. With a more capable setup and time to test the networks, there is the opportunity to create more intensive networks. As well as this larger batch sizes can be used for more stable training with higher RAM available.
2. Compare the performance of the reconstructed pipeline like-for-like against a video stream from a frame-based camera. This has been omitted from this project due to time constraints and lack of available datasets.
3. Change E2VID parameters so that resulting reconstructed video is of a higher framerate.
4. Test the performance of the described networks and pipelines on more readily available datasets to make sure that the findings apply to other environments. Intensity reconstructions may be more beneficial for more complex tasks.
5. Test performance gain with custom frame integration method with all networks.
6. Look more at the use of natural language processing techniques when working with event streams.
7. Create more spiking neural networks, perhaps from scratch rather than converting regular ANNs. Further testing needs to be done in this area.
8. Test the performance of the end-to-end event classification networks when frame-integration is done asynchronously as well as synchronously (as described in Section 2.3.2). Comparisons between the two forms of integration, as well as effect of having smaller time-slices per frame need to be tested.

9. Create spiking neural networks based on more complex ANN networks. This involves creating approximations to other, more complex layers such as LSTM and GRU layers.
10. Implement spiking neural networks on neuromorphic hardware. This also opens up the possibility of real-time inference.
11. Create a spiking version of the intensity reconstruction algorithms to check if they perform well.

6.2 Similar External Work

A method, similar to the custom frame integration proposed in Section 4.3, that aimed to preserve temporal resolution was proposed by Loïc Cordone *et al.*, who proposed 'voxel cubes'[42]. For this method there were still binary events in every channel, but there were more than two channels so that the events in each time-slice could be subdivided into each channel. When compared to this method, the temporal information can be stored in the same way with the novel method proposed in this project, while keeping data size small since it is just a one-channel image.

- TODO: Look at these: `https://arxiv.org/pdf/2205.04339.pdf` and `https://arxiv.org/pdf/2104.12579.pdf` COULD MEAN WORK IS NOT STATE OF THE ART UGHHHH.
- `https://arxiv.org/pdf/2205.04339.pdf` using channel dimension could be future work.

Appendix A

Classification Model Architectures

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_10 (Conv3D)           (None, 8, 34, 34, 32)     4032

batch_normalization_12 (Bat  (None, 8, 34, 34, 32)     128
chNormalization)

conv3d_11 (Conv3D)           (None, 8, 34, 34, 32)     128032

batch_normalization_13 (Bat  (None, 8, 34, 34, 32)     128
chNormalization)

max_pooling3d_6 (MaxPooling  (None, 4, 17, 17, 32)     0
3D)

dropout_8 (Dropout)          (None, 4, 17, 17, 32)     0

conv3d_12 (Conv3D)           (None, 4, 17, 17, 64)     256064

batch_normalization_14 (Bat  (None, 4, 17, 17, 64)     256
chNormalization)

max_pooling3d_7 (MaxPooling  (None, 2, 9, 9, 64)       0
3D)

dropout_9 (Dropout)          (None, 2, 9, 9, 64)       0

conv3d_13 (Conv3D)           (None, 2, 9, 9, 128)      1024128

batch_normalization_15 (Bat  (None, 2, 9, 9, 128)      512
chNormalization)

conv3d_14 (Conv3D)           (None, 2, 9, 9, 128)      2048128

batch_normalization_16 (Bat  (None, 2, 9, 9, 128)      512
chNormalization)

max_pooling3d_8 (MaxPooling  (None, 1, 5, 5, 128)      0
3D)

dropout_10 (Dropout)         (None, 1, 5, 5, 128)      0

flatten_2 (Flatten)          (None, 3200)              0

dense_4 (Dense)              (None, 128)               409728

batch_normalization_17 (Bat  (None, 128)               512
chNormalization)

dropout_11 (Dropout)         (None, 128)               0

dense_5 (Dense)              (None, 10)                1290

activation_2 (Activation)    (None, 10)                0

=================================================================
Total params: 3,873,450
Trainable params: 3,872,426
Non-trainable params: 1,024
_________________________________________________________________
```

Listing A.1: Overview of layers in 3D convolutional network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lstm2d_6 (ConvLSTM2D)   (None, 8, 34, 34, 64)     150016

max_pooling3d_4 (MaxPooling  (None, 8, 17, 17, 64)     0
3D)

batch_normalization_6 (Batc  (None, 8, 17, 17, 64)     256
hNormalization)

conv_lstm2d_7 (ConvLSTM2D)   (None, 8, 17, 17, 32)     110720

max_pooling3d_5 (MaxPooling  (None, 8, 9, 9, 32)       0
3D)

batch_normalization_7 (Batc  (None, 8, 9, 9, 32)       128
hNormalization)

conv_lstm2d_8 (ConvLSTM2D)   (None, 9, 9, 16)          27712

max_pooling2d_2 (MaxPooling  (None, 5, 5, 16)          0
2D)

batch_normalization_8 (Batc  (None, 5, 5, 16)          64
hNormalization)

flatten_2 (Flatten)          (None, 400)               0

dense_4 (Dense)              (None, 256)               102656

dense_5 (Dense)              (None, 10)                2570

=================================================================
Total params: 394,122
Trainable params: 393,898
Non-trainable params: 224
_________________________________________________________________
```

Listing A.2: Overview of layers in Convolutional LTSM network.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_3 (TimeDis  (None, 8, 2048)          4887936
tributed)

gru_3 (GRU)                  (None, 64)                405888

dense_25 (Dense)             (None, 1024)              66560

dropout_9 (Dropout)          (None, 1024)              0

dense_26 (Dense)             (None, 512)               524800

dropout_10 (Dropout)         (None, 512)               0

dense_27 (Dense)             (None, 128)               65664

dropout_11 (Dropout)         (None, 128)               0

dense_28 (Dense)             (None, 64)                8256

dense_29 (Dense)             (None, 10)                650

=================================================================
Total params: 5,959,754
Trainable params: 5,959,754
Non-trainable params: 0
_________________________________________________________________
```

Listing A.3: Overview of layers in Custom Convolutional LTSM network.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 34, 34, 128)       640

activation_3 (Activation)    (None, 34, 34, 128)       0

max_pooling2d_4 (MaxPooling  (None, 17, 17, 128)       0
2D)

conv2d_4 (Conv2D)            (None, 17, 17, 256)       131328

activation_4 (Activation)    (None, 17, 17, 256)       0

max_pooling2d_5 (MaxPooling  (None, 8, 8, 256)         0
2D)

conv2d_5 (Conv2D)            (None, 8, 8, 512)         524800

activation_5 (Activation)    (None, 8, 8, 512)         0

max_pooling2d_6 (MaxPooling  (None, 4, 4, 512)         0
2D)

flatten_2 (Flatten)          (None, 8192)              0

dense_7 (Dense)              (None, 256)               2097408

activation_6 (Activation)    (None, 256)               0

dense_8 (Dense)              (None, 10)                2570

activation_7 (Activation)    (None, 10)                0

=================================================================
Total params: 2,756,746
Trainable params: 2,756,746
Non-trainable params: 0
_________________________________________________________________
```

Listing A.4: Overview of layers in Custom 2D Convolutional network built into the Custom LSTM Network in listing A.3.

Appendix B

Evaluation Metrics of Each Trained Model

	Precision	Recall	F1-score	Samples
0	1.00	0.99	0.99	500
1	0.99	1.00	0.99	500
2	0.99	0.99	0.99	500
3	0.98	1.00	0.99	500
4	0.99	0.99	0.99	500
5	0.99	0.97	0.98	500
6	0.99	0.99	0.99	500
7	0.99	0.97	0.98	500
8	0.97	0.99	0.98	500
9	0.98	0.98	0.98	500
macro avg	0.99	0.99	0.99	5000
weighted avg	0.99	0.99	0.99	5000

Table B.1: A table showing classification evaluation metrics of 3D convolutional network on the frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.97	1.00	0.98	500
1	0.99	0.99	0.99	500
2	0.99	0.99	0.99	500
3	0.99	0.96	0.97	500
4	0.99	0.95	0.97	500
5	0.97	0.98	0.98	500
6	0.99	0.97	0.98	500
7	0.97	0.98	0.97	500
8	0.96	0.97	0.97	500
9	0.94	0.98	0.96	500
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

Table B.2: A table showing classification evaluation metrics of convolutional LSTM network on the frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.98	0.99	0.99	500
1	0.98	0.99	0.99	500
2	0.99	0.97	0.98	500
3	0.99	0.96	0.97	500
4	0.99	0.98	0.98	500
5	0.98	0.97	0.97	500
6	0.98	0.99	0.98	500
7	0.96	0.99	0.98	500
8	0.94	0.97	0.95	500
9	0.98	0.96	0.97	500
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

Table B.3: A table showing classification evaluation metrics of custom convolutional LSTM network on the frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	1.00	0.97	0.98	500
1	0.99	1.00	0.99	500
2	0.99	0.99	0.99	500
3	0.99	0.99	0.99	500
4	0.99	0.97	0.98	500
5	1.00	0.98	0.99	500
6	0.97	1.00	0.98	500
7	0.98	0.97	0.98	500
8	0.97	0.97	0.97	500
9	0.95	0.98	0.96	500
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

Table B.4: A table showing classification evaluation metrics of 3D convolutional network on the custom frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.99	0.99	0.99	500
1	0.98	1.00	0.99	500
2	0.97	0.99	0.98	500
3	0.94	0.99	0.96	500
4	1.00	0.98	0.99	500
5	0.96	0.99	0.98	500
6	0.99	0.99	0.99	500
7	0.96	0.98	0.97	500
8	0.98	0.91	0.95	500
9	0.97	0.94	0.96	500
macro avg	0.98	0.97	0.97	5000
weighted avg	0.98	0.97	0.97	5000

Table B.5: A table showing classification evaluation metrics of convolutional LSTM network on the custom frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.98	0.98	0.98	500
1	0.98	1.00	0.99	500
2	0.98	0.99	0.98	500
3	0.98	0.97	0.97	500
4	0.99	0.98	0.98	500
5	0.98	0.98	0.98	500
6	0.97	0.99	0.98	500
7	0.96	0.98	0.97	500
8	0.97	0.94	0.96	500
9	0.98	0.94	0.96	500
macro avg	0.98	0.98	0.98	5000
weighted avg	0.98	0.98	0.98	5000

Table B.6: A table showing classification evaluation metrics of custom convolutional LSTM network on the custom frame-integrated NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.94	0.96	0.95	75
1	0.95	0.96	0.95	75
2	0.85	0.93	0.89	75
3	0.84	0.71	0.77	75
4	0.93	0.84	0.88	75
5	0.71	0.87	0.78	75
6	0.87	0.91	0.89	75
7	0.91	0.85	0.88	75
8	0.86	0.73	0.79	75
9	0.85	0.91	0.88	75
macro avg	0.87	0.87	0.87	750
weighted avg	0.87	0.87	0.87	750

Table B.7: A table showing classification evaluation metrics of 3D convolutional network on the reconstructed NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.83	0.92	0.87	75
1	0.96	0.96	0.96	75
2	0.82	0.91	0.86	75
3	0.77	0.57	0.66	75
4	0.74	0.72	0.73	75
5	0.67	0.71	0.69	75
6	0.88	0.85	0.86	75
7	0.85	0.84	0.85	75
8	0.74	0.64	0.69	75
9	0.71	0.84	0.77	75
macro avg	0.80	0.80	0.79	750
weighted avg	0.80	0.80	0.79	750

Table B.8: A table showing classification evaluation metrics of convolutional LSTM network on the reconstructed NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.88	0.95	0.91	75
1	0.99	0.92	0.95	75
2	0.93	0.85	0.89	75
3	0.79	0.76	0.78	75
4	0.96	0.73	0.83	75
5	0.71	0.81	0.76	75
6	0.81	0.92	0.86	75
7	0.82	0.84	0.83	75
8	0.68	0.68	0.68	75
9	0.82	0.85	0.84	75
macro avg	0.84	0.83	0.83	750
weighted avg	0.84	0.83	0.83	750

Table B.9: A table showing classification evaluation metrics of custom convolutional LSTM network on the reconstructed NMNIST dataset.

	Precision	Recall	F1-score	Samples
0	0.55	0.88	0.68	24
1	0.83	1.00	0.91	24
2	0.77	0.83	0.80	24
3	0.80	1.00	0.89	24
4	0.60	0.75	0.67	24
5	0.75	0.38	0.50	24
6	0.80	0.33	0.47	24
7	0.57	0.88	0.69	24
8	0.75	0.79	0.77	48
9	0.89	0.33	0.48	24
10	0.94	0.62	0.75	24
macro avg	0.75	0.71	0.69	288
weighted avg	0.75	0.72	0.70	288

Table B.10: A table showing classification evaluation metrics of 3D convolutional network on the frame-integrated DVS128 Gesture dataset.

	Precision	Recall	F1-score	Samples
0	0.55	0.88	0.68	24
1	0.83	1.00	0.91	24
2	0.77	0.83	0.80	24
3	0.80	1.00	0.89	24
4	0.60	0.75	0.67	24
5	0.75	0.38	0.50	24
6	0.80	0.33	0.47	24
7	0.57	0.88	0.69	24
8	0.75	0.79	0.77	48
9	0.89	0.33	0.48	24
10	0.94	0.62	0.75	24
macro avg	0.75	0.71	0.69	288
weighted avg	0.75	0.72	0.70	288

Table B.11: A table showing classification evaluation metrics of convolutional LSTM network on the frame-integrated DVS128 Gesture dataset.

	Precision	Recall	F1-score	Samples
0	0.77	0.83	0.80	24
1	0.92	0.96	0.94	24
2	0.82	0.75	0.78	24
3	0.91	0.88	0.89	24
4	0.58	0.75	0.65	24
5	0.70	0.58	0.64	24
6	0.52	0.58	0.55	24
7	0.54	0.54	0.54	24
8	0.94	0.98	0.96	48
9	0.70	0.67	0.68	24
10	0.94	0.67	0.78	24
macro avg	0.76	0.74	0.75	288
weighted avg	0.77	0.76	0.76	288

Table B.12: A table showing classification evaluation metrics of a convolutional LSTM on the frame-integratedDVS128 Gesture dataset.

	Precision	Recall	F1-score	Samples
0	0.53	0.88	0.66	24
1	0.85	0.96	0.90	24
2	0.88	0.88	0.88	24
3	0.43	0.92	0.59	24
4	0.42	0.21	0.28	24
5	0.61	0.58	0.60	24
6	0.52	0.71	0.60	24
7	0.76	0.60	0.67	48
8	1.00	0.21	0.34	24
9	0.91	0.42	0.57	24
macro avg	0.69	0.64	0.61	264
weighted avg	0.70	0.63	0.61	264

Table B.13: A table showing classification evaluation metrics of 3D convolutional network on reconstructed DVS128 Gesture dataset.

	Precision	Recall	F1-score	Samples
0	0.67	0.83	0.74	24
1	0.86	1.00	0.92	24
2	0.89	1.00	0.94	24
3	0.85	0.46	0.59	24
4	0.66	0.88	0.75	24
5	0.95	0.79	0.86	24
6	0.81	0.88	0.84	24
7	0.88	0.92	0.90	48
8	0.81	0.54	0.65	24
9	1.00	0.92	0.96	24
macro avg	0.84	0.82	0.82	264
weighted avg	0.84	0.83	0.82	264

Table B.14: A table showing classification evaluation metrics of custom convolutional LSTM network on reconstructed DVS128 Gesture dataset.

Bibliography

[1] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[2] X. Liang, X. Zhang, J. Xia, M. Ezawa, Y. Zhao, G. Zhao, and Y. Zhou, "A spiking neuron constructed by the skyrmion-based spin torque nano-oscillator," *Applied Physics Letters*, vol. 116, no. 12, p. 122402, 2020.

[3] Q. Miao, Y. Li, W. Ouyang, Z. Ma, X. Xu, W. Shi, and X. Cao, "Multimodal gesture recognition based on the resc3d network," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 3047–3055.

[4] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-video: Bringing modern computer vision to event cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3857–3866.

[5] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[6] ——, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[7] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017.

[8] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017.

[9] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre memory units: Continuous-time representation in recurrent neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[10] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

[11] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 $\times$ 128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[12] N. Perez-Nieves, V. C. Leung, P. L. Dragotti, and D. F. Goodman, "Neural heterogeneity promotes robust learning," *bioRxiv*, pp. 2020–12, 2021.

[13] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *arXiv preprint arXiv:2103.12593*, 2021.

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

[15] E. Hunsberger and C. Eliasmith, "Training spiking deep networks for neuromorphic hardware," *arXiv preprint arXiv:1611.05141*, 2016.

[16] G. Gallego, C. Forster, E. Mueggler, and D. Scaramuzza, "Event-based camera pose tracking using a generative event model," *arXiv preprint arXiv:1510.01972*, 2015.

[17] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2661–2671.

[18] W. Fang, Y. Chen, J. Ding, D. Chen, Z. Yu, H. Zhou, Y. Tian, and other contributors, "Spikingjelly," https://github.com/fangwei123456/spikingjelly, 2020, accessed: 2022-05-20.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[20] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "Hots: a hierarchy of event-based time-surfaces for pattern recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1346–1359, 2016.

[21] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 491–501.

[22] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[25] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[28] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[29] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021. [Online]. Available: https://www.R-project.org/

[30] B. Stroustrup, *The C++ programming language*, 3rd ed. Addison-Wesley, 1997.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[33] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[34] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, vol. 17, no. 4, pp. 611–628, 2019.

[35] Google, "Google colaboratory," https://colab.research.google.com/, 2017, accessed: 2022-05-20.

[36] Amazon, "Aws sagemaker," https://aws.amazon.com/sagemaker/, 2017, accessed: 2022-05-20.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] M. R. Antoine Cully and J. Wang, "Introduction to machine learning (co395), lecture 3," University Lecture, 2020.

[39] Nengo, "Legendre memory units in nengodl," https://www.nengo.ai/nengo-dl/examples/lmu.html, accessed: 2022-06-15.

[40] R. F. Pinto, C. D. Borges, A. Almeida, and I. C. Paula, "Static hand gesture recognition based on convolutional neural networks," *Journal of Electrical and Computer Engineering*, vol. 2019, 2019.

[41] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[42] L. Cordone, B. Miramond, and P. Thierion, "Object detection with spiking neural networks on automotive event data," *arXiv preprint arXiv:2205.04339*, 2022.