MEng Individual Project

Imperial College London

Department of Electrical and Electronic Engineering

Event Analysis for Classification of Neuropmorhic Data

Author:
Tejas Dandawate

Supervisor:
Prof. Pier Luigi Dragotti

Second Marker:
Prof. Patrick A. Naylor

June 1, 2022

Acknowledgements

Thanks mum!

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 they 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. The many purported benefits of 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 in built into the system, this can be seen as a stop-gap measure.

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 Objectives

The objectives of this project were roughly divided into two main sections; main objectives and extensions. Objectives were created so as to allow for contingencies and fallbacks at every stage.

1.2.1 Main Objectives

- Understand structure of neuromorphic data and methods to preprocess it for inputting into an Artificial Neural Network (ANN).
- Evaluate different ANN models on neuromorphic data from external datasets.

  This allows for determining and comparing the performance of both traditional Neural Networks (NNs) as well of Spiking Neural Networks (SNNs). These networks can take two main forms:

  - Two networks sequentially processing data. The first would be reconstructing spiking data into an intensity video, and the second would be applying pre-existing computer vision networks to analyse the frame-based output.
  - One network that takes the spiking data as input to directly carry out the intended function.

  The system will initially be used to solve a classification task.

- Create practical set-up to experimentally record data.

  Using a neuromorphic camera, data can be experimentally captured to assess the performance of any built networks on more realistic unseen input data.

- Carry out Simultaneous Localisation and Mapping (SLAM) for a robot moving in an unknown trajectory.

  Use most efficient NN model to solve the more complex task of SLAM. The neuromorphic camera can be used to capture data from a room in the absence of large sets of labelled datasets.

1.2.2 Challenges & Contingency Plan

- Only using existing datasets rather than practical set-up.

  Since there is sufficient amounts of existing datasets for classification tasks, they can be split into training, validation, and testing sets themselves. This eliminates the need to generate more data for the testing process.

- Rather than focusing on SLAM the emphasis may be on object detection/recognition or gesture recognition

  The project has been segmented into individual milestones, and so even if the final milestone of a SLAM algorithm isn't achieved, it is easy to change the scope of the project to be a classification or gesture recognition network.

1.3 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)

Implementation (Chapter 4)

This chapter outlines some initial steps taken in the implementation process.

Testing and Results (Chapter 5)

Conclusion and Further Work (Chapter 7)

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)**.

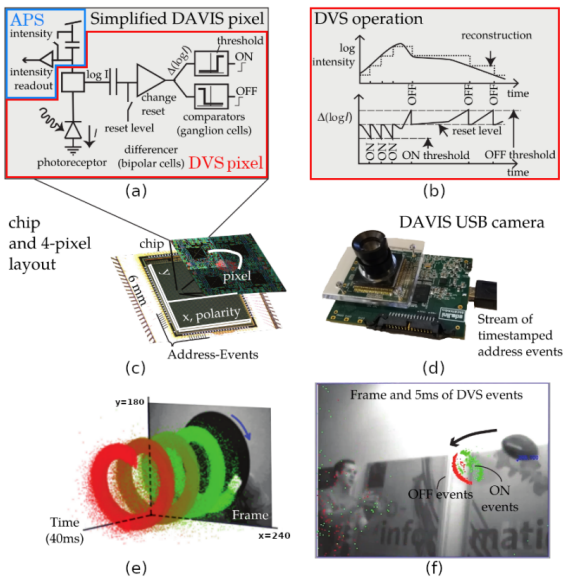


Figure 2.1: A summary of the functionality of the DAVIS event based camera[1].

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.*[10], 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[11], 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.

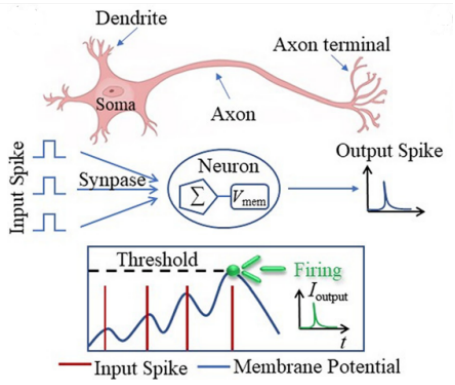


Figure 2.2: A diagram of structure and function of a biological neuron [2].

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.*[12] 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.

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)[13]. 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.

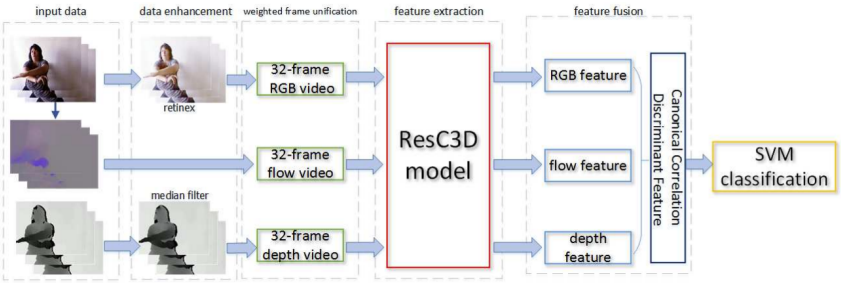


Figure 2.3: A diagram showing the pipeline for a multimodal gesture recognition system[3].

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.*[14], which is the one used in the python package spikingjelly[15]. 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'[16] 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.

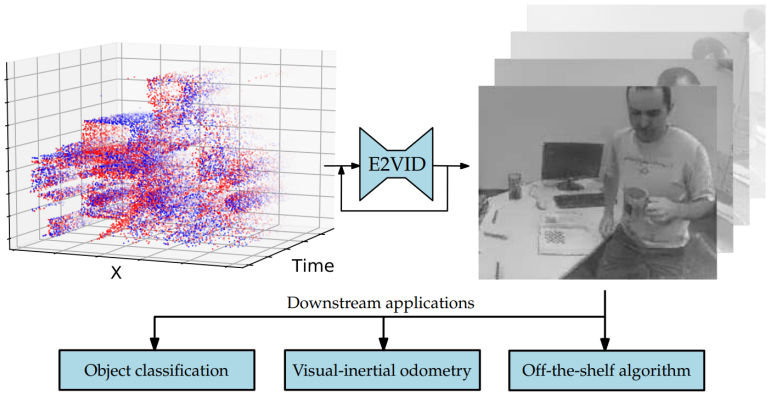


Figure 2.4: An illustration of the mapping of spiking data to video stream to apply off-the-shelf algorithms to[4].

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.

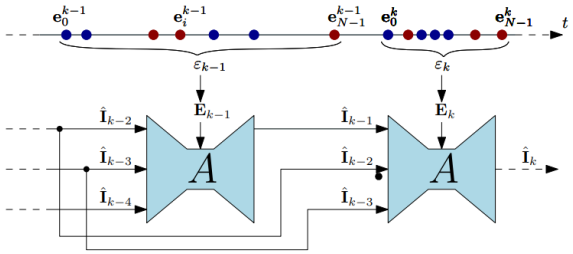


Figure 2.5: An overview of RNN used to generate video from sets of events[4].

2.4.1 End-to-end Event Classification Network

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).

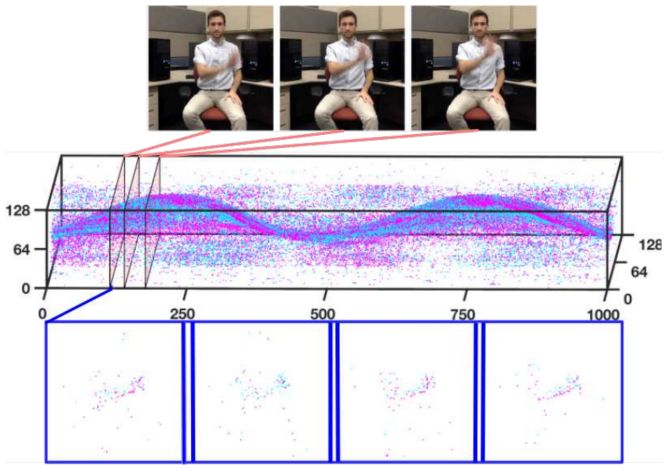


Figure 2.6: An illustration of a temporal filter that caches events fired from a camera[5].

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.*[17] moves towards building time surfaces from events to feed into a neural network, whereas Yin Bi *et al.*[18] propose using a non-uniform sampler to create a graph from events to then feed into a network of so-called graph convolution networks.

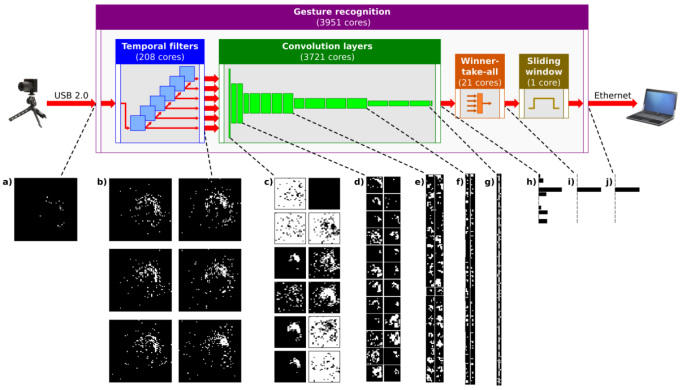


Figure 2.7: A network designed to perform gesture recognition using neuromorphic input by making use of cascading temporal filters[5].

2.5 Temporally Aware Deep Learning and Classification Models

2.5.1 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 [19]. 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.

2.5.2 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[20] 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.

2.5.3 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[21]. 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.

2.5.4 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.*, 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.

TODO: Write more about convLTSM networks etc.

2.6 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.

2.6.1 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].

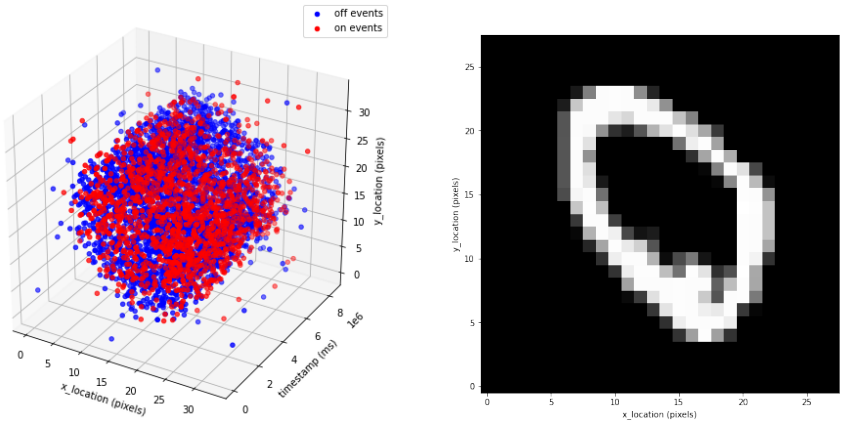
NMNIST

This dataset is a spiking version of the original frame-based MNIST dataset [22][9]. 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.

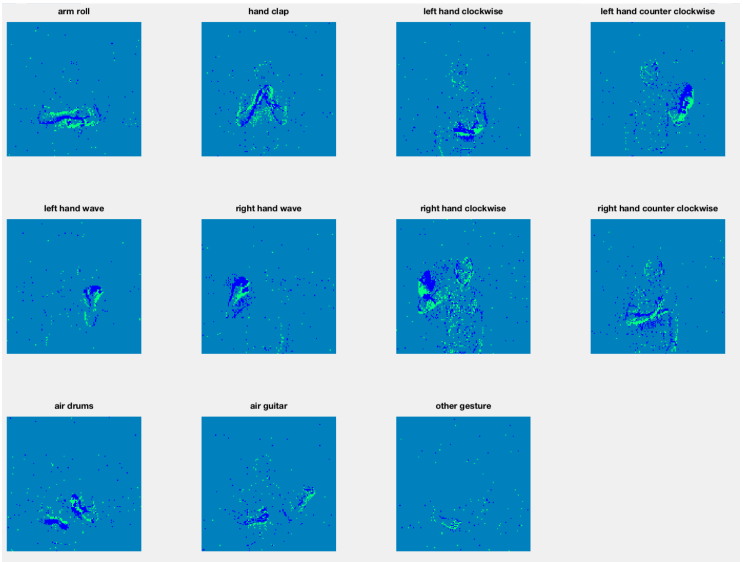


Figure 2.10: Visualisation of all gestures captured in the DVS128 Gesture dataset[6].

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.

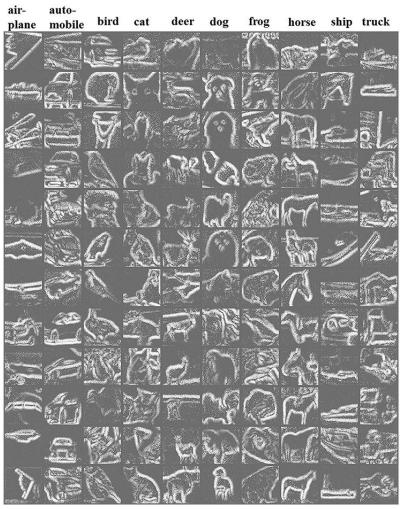


Figure 2.11: Visualisation of all objects captured in the CIFAR10-DVS dataset[7].

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.

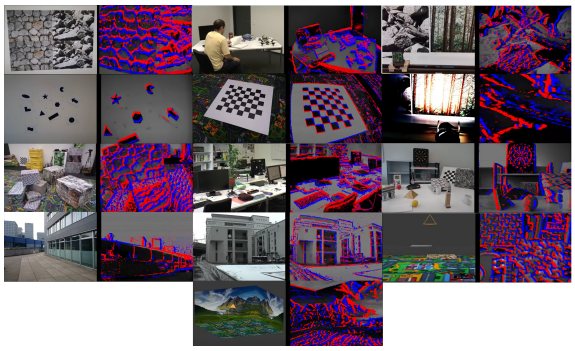


Figure 2.12: Visualisation of all videos and events captured in the Event-Camera dataset[8].

Heridelberg Spiking Datasets

The Spiking Heidelberg datasets for spiking neural networks[23] 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.

2.6.2 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.*[11]. This avoids having to own an event-based camera to create data (as was done with the NMNIST dataset mentioned in section 2.6.1), 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 form the basis of each decision. The design of the project is mostly guided by the requirements set out at the start of the chapter. The choice programming language and 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. The final part of this chapter also creates a framework for the comparison of each network, which can be found in chapter 5 and chapter 6.

3.1 Requirements Capture

TODO: Write out a list of requirements for the project.

3.2 Hardware and Software

3.2.1 Programming Languages

When choosing a programming language for the project there were a few choices that are most often chosen by developers; Python[24], R[25], and C++[26]. 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.

3.2.2 Machine Learning Frameworks and Software

PyTorch

Pytorch[27] is a relatively new framework for machine learning, and provides a developer friendly way to write machine learning code. It is a more 'pythonic' approach to code abstraction that its competition in the space, and the *torch.nn.module* gives access to clear, reusable module definitions in an Object-Oriented Programming 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 itt can use standard debugging tools such as PyCharm and pdb.

Tensorflow

Tensorflow[28] 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[29] is a framework developed by Google, and uses a primarily Tensorflow based back-end. It provides an easy to use API for fast prototyping abd 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.

3.2.3 Other Software

SpikingJelly

SpikingJelly[15] 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 2.6), as well as a simple set of classes for building SNNs or converting ANNs to SNNs. The library, which was mainly co-developed by Multimedia Learning Group, Institute of Digital Media (NELVT), 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[30] 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 these 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.

TODO: This could be moved if Nengo doesn't end up being used.

3.2.4 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 means that code runs much faster on more powerful machines with this specialised hardware in them, which was not the case for hardware readily available during the course of the project. For this reason cloud services provided by the likes of Google and Amazon Web Services are a good alternative to physically owning hardware. 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[31] and AWS Sagemaker Studio Lab[32]. In terms of hardware, both services offer access to great 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.

3.3 Datasets

The datasets used for the evaluation of the networks in this project were NMNIST (Section 2.6.1), DVS128 Gesture (Section 2.6.1) and CIFAR10-DVS (Section 2.6.1), 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 2.6.1) are also extensive and include side by side frames from a traditional camera, and so were ideal to use for testing the event reconstruction algorithms.

3.3.1 Data Pre-processing

Before processing it, 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, 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 of the training dataset, since it is important to never use values from the testing set since that would invalidate them as true indicators of the networks performance.

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

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 on pixel respectively. 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 can also be captured. A diagram showing the basic process with this custom integration method is shown in fig. 3.1.

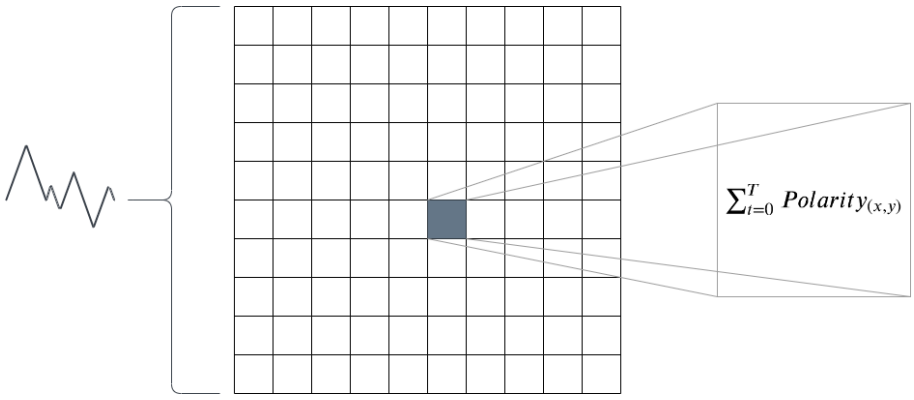


Figure 3.1: An illustration of the frame integration process with custom integrating method.

TODO: Maybe write more about this and add more stuff about synchronous/asynchronous frame integration.

3.5 Two-phase Intensity Reconstruction Models

In order to make use of existing highly-tested and documented computer vision techniques, intensity reconstruction models were first applied to events. For this reason a two-phase network (shown in fig. 3.2) was devised, wherein events are reconstructed into intensity videos before being passed into various classification networks..

TODO: Add system diagram of various networks being fed by E2VID and explain it.

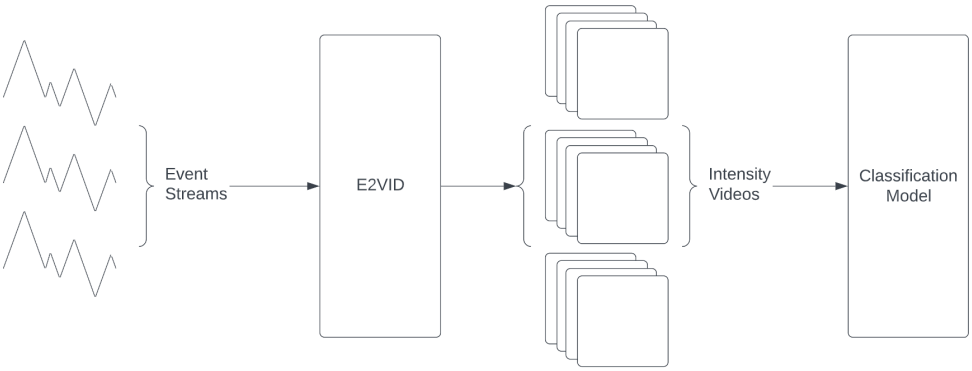


Figure 3.2: An illustration of the two-phase classification pipeline.

3.6 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, which 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 possible it can rise again. This, of course, means that the training data-set needs to be split into training and validation, where the validation set is more unseen data to evaluate the performance of the networks per epoch.

As well as this 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.

3.6.1 3-Dimensional Convolutional Neural Network

A 3D convolutional network as described in section 2.5.1 was implemented to work directly on the event data. TODO: Write more about this network.

3.6.2 Convolutional LSTM Network

A convolutional LSTM network as described in section 2.5.4 was implemented to work directly on the event data. TODO: Write more about this network.

3.6.3 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. 3.3. This additional capacity is helpful for deciphering more complex 'actions' in a video, such as in the case of gesture recognition.

3.7 Evaluation Metrics

The evaluation plan is as given by the typical machine learning pipeline[33]. TODO: Change this to actual used evaluation metrics and check it fits properly in background

For a classification task, when we obtain the results from the test dataset (as shown in table 3.1) we can calculate a variety of evaluation metrics that give various insights on our final model.

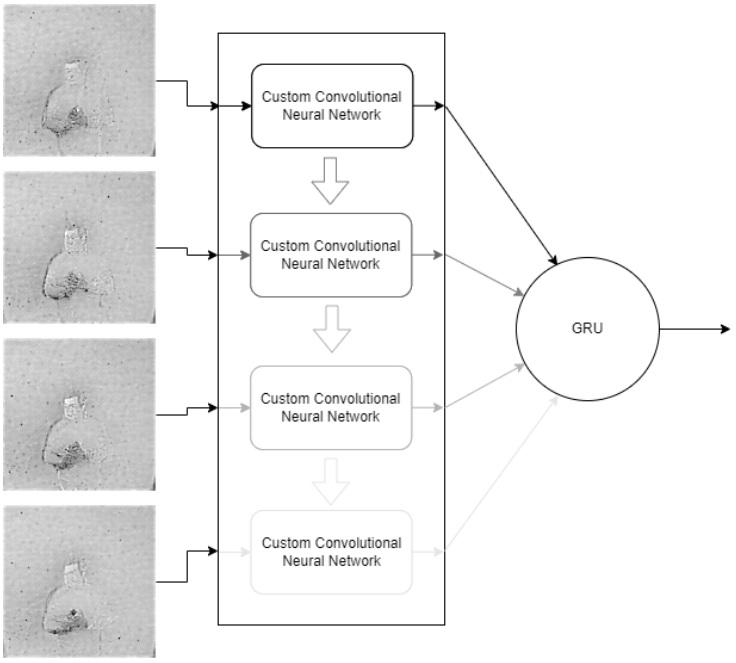


Figure 3.3: An illustration of the custom convolutional LSTM network.

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[9] into the final model.

3.7.1 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 3.7.6). 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).

3.7.2 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 $1 - accuracy$.

3.7.3 Precision

Precision is the proportion of positively predicted samples identified correctly.

		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[9] for class 1 as the positive class.

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

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

3.7.4 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.

3.7.5 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.

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

3.7.6 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 End-to-end Event Classification Models

Frame Integration

In order to begin analysing neuromorphic data, it was pre-processed it into a form that a NN can take as input. One such method of doing so was to segment the events into groups based on their timestamp. Figure 4.1 shows a visualisation of intensity maps created from the NMNIST[9] 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. It was clear that the resulting output greatly resembled the MNIST[22] sample recorded by the ATIS camera (As shown in fig. 2.9 in section 2.6). This is an implementation of the more formal method of processing events streams to get integrated frames is given in section 2.3.2. This technique was also applied to two other readily available data-sets; DVS128 Gesture and CIFAR10-DVS.

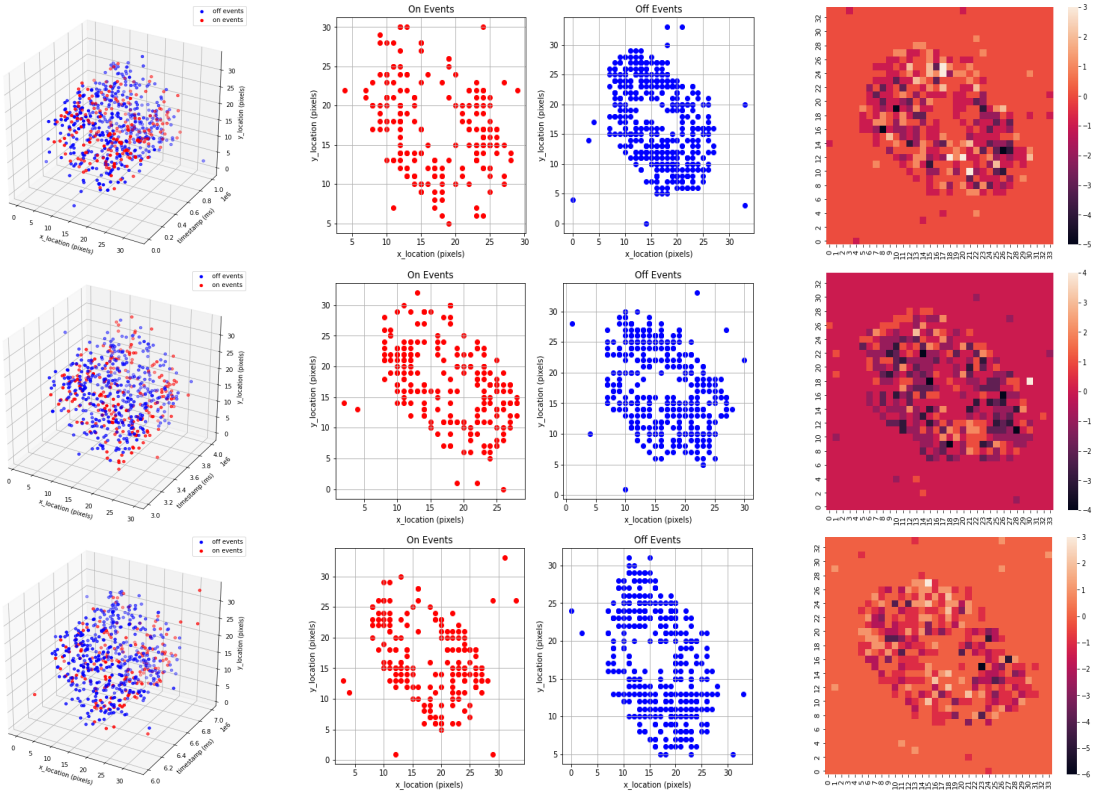
The integrated frames of a hand gesture in fig. 4.2 shows the motion of an arm moving in a clockwise direction. For each instance of a gesture the event stream was split into 20 frames. This made the processing of the data easy for some of the networks described later in the chapter, since the frames could be packed into 3D tensors of equal size. This means that the frames are not synchronous like they would be from an 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 easily distinguishable features may be seen. 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.

TODO: Read over this and make sure that the information in it is not repeated from the previous paragraph.

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

TODO: Include image and explanations of frame integration

TODO: Add stuff about possible nlp-like networks.



(a) (b) (c)

Figure 4.1: A visualisation of intensity maps created by segmenting events into bins of size 1×10^6 ms.

4.2 Two-phase Intensity Reconstruction Models

The models described above allow for the system to learn directly on the spiking data. Another approach that was taken was to attempt to utilise video reconstruction networks to the event data so that more classical models and architectures could be used to patterns in the data.

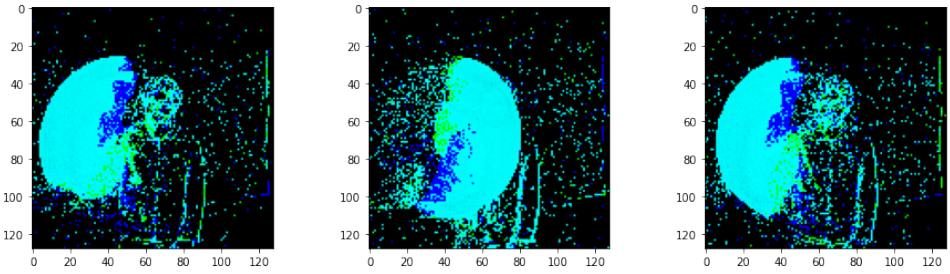
4.2.1 Reconstruction Algorithms

E2VID

E2VID, as described in TODO: Reference background reading here , is a state-of the art network based on UNET that reconstructs intensity videos from events data. Having gotten the output from the network (which on test data had 90% accuracy TODO: check accuracy figure), 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 which had fewer frames than that of the longest video in the training set. These additional frames were added by repeating the video until the desired number of frames was reached.

TODO: all of the following;

- Similar to denoising network CNN more easily able to find patterns in data
- gesture intensity maps equivalent to 100fps but retains high frequency information without too much noise
- spikingjelly for integrating frames which is widely used approach



(a) (b) (c)

Figure 4.2: Three contiguous intensity frames (a, b and c), created from the DVS128 Gesture dataset with a person moving their right hand clockwise.

4.3 Classification Models

The models used to classify the either the integrated frames or 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).

4.3.1 3D Convolutional Neural Network

The 3D convolutional neural network in section 2.5.1 was altered to have the outputs of the hidden layer fed into a dense layer with 10 neurons to classify the correct class (0-9). Activation functions need to be present in the network to prevent all the layers from becoming equivalent to a single one TODO: add references here (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), and in this network as well, is ReLU. TODO: add references here . Finally, the output layer has 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 10 classes. This means we can simply take the highest probability as the predicted class from the network.

Hyper-parameter Tuning:

In order to choose the most appropriate parameters for the system, 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 3.7, 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.

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.

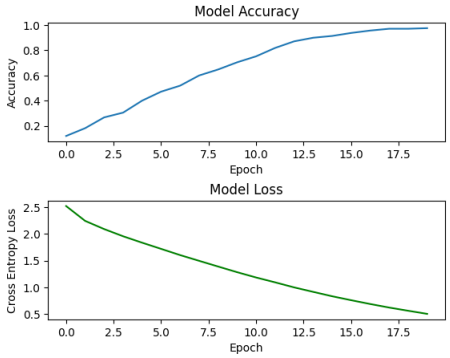
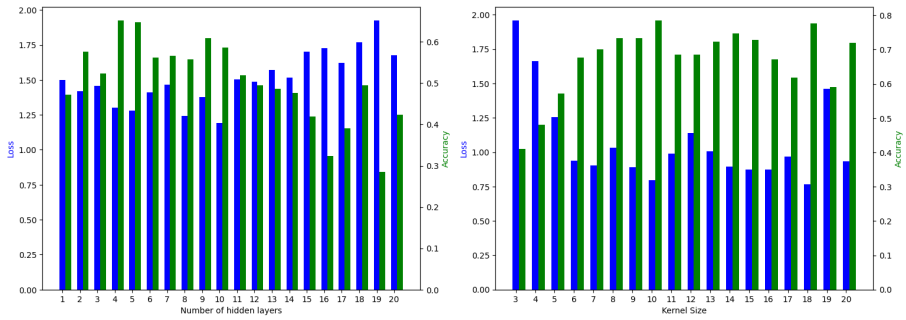


Figure 4.3: A figure showing classification accuracy and cross-entropy loss per epoch on training data for a typical network.

Model capacity is directly correlated to the n^o 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 value 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 As for the dense layers at the end of the network, it can be seen that better results were achieved as a result of it since global patterns can be further identified after the data has been processed by the convolution layers that have picked out the most important features.



(a) (b)

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

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

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 4.1. 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 2D 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. 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.

4.3.2 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 4.2. 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 2.5.4.

4.3.3 Custom Convolutional LSTM

An implementation of a custom convolutional LSTM can be seen in listing 4.3 and the implementation of the custom 2D convolutional network applied to each frame of the video can be seen in listing 4.4. This network is the natural progression from the previous convLSTM network since it applies a more complex 2D convolution 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 4.2.

4.3.4 Spiking Neural Network

TODO: Create design for spiking network with nengo.

Synaptic Smoothing

Firing Rates

Post-training Scaling

Regularizing During Training

```
_________________________________________________________________
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 4.1: 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 4.2: 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 4.3: Overview of layers in Custom Convolutional LTSM network.

```
_________________________________________________________________
    Layer (type)                 Output Shape              Param #
=================================================================
    conv2d_4 (Conv2D)            (None, 8, 34, 512)        157184

    activation_5 (Activation)    (None, 8, 34, 512)        0

    max_pooling2d_3 (MaxPooling  (None, 4, 17, 512)        0
    2D)

    conv2d_5 (Conv2D)            (None, 4, 17, 256)        3277056

    activation_6 (Activation)    (None, 4, 17, 256)        0

    max_pooling2d_4 (MaxPooling  (None, 2, 8, 256)         0
    2D)

    conv2d_6 (Conv2D)            (None, 2, 8, 128)         1605760

    activation_7 (Activation)    (None, 2, 8, 128)         0

    max_pooling2d_5 (MaxPooling  (None, 1, 4, 128)         0
    2D)

    flatten_1 (Flatten)          (None, 512)               0

    dense_2 (Dense)              (None, 256)               131328

    activation_8 (Activation)    (None, 256)               0

    dense_3 (Dense)              (None, 10)                2570

    activation_9 (Activation)    (None, 10)                0

=================================================================
Total params: 5,173,898
Trainable params: 5,173,898
Non-trainable params: 0
_________________________________________________________________
```

Listing 4.4: Overview of layers in Custom 2D Convolutional network built into the Custom LSTM Network in listing 4.3.

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 were previously outlines in section 3.7, allowing 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 3.3.1, 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 End-to-end Event Classification Models

The benefits of the event driven camera (as described in section 2.1.1) were evident in the acquired results. TODO: Add images of integrated frames here . 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).

5.2.1 Frame Integration

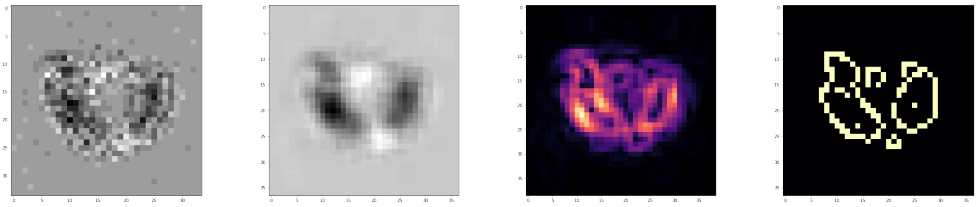
It is, however, evident that modern computer vision techniques have been developed with frame-based cameras in mind, and so modern networks achieve good accuracy, and are able to find patterns well, on frame-based data. For this reason the common technique of integrating frame TODO: Add reference to spikingjelly integrating frames from background reading here results in regular frames, akin to the ones a regular camera generates, in order to feed into such networks. 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 TODO: Add references to common gesture recognition architectures here for classical videos feature an intermediate layer to remove backgrounds and other noise from images from the image to focus on the points of interest. In this way the intermediate layer could be omitted when operating on the integrated event frames, since the output was already similar to an edge map TODO: Add image comparison here . It is conceivable, however, that in noisy environments with lots of motion this stage would still be necessary.

It was also interesting to note the inherent ability of neuromorphic cameras to capture points of interest. 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 5.1 shows the effect of carrying out canny edge detection[34] 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 5.1: 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.

TODO: Write about the two different frame integration methods and show diagrams.
TODO: Write about the performance difference between synchronous and asynchronous frame integration

5.3 Two-phase Intensity Reconstruction Models

TODO: Write a nice intro to this section like the last section.

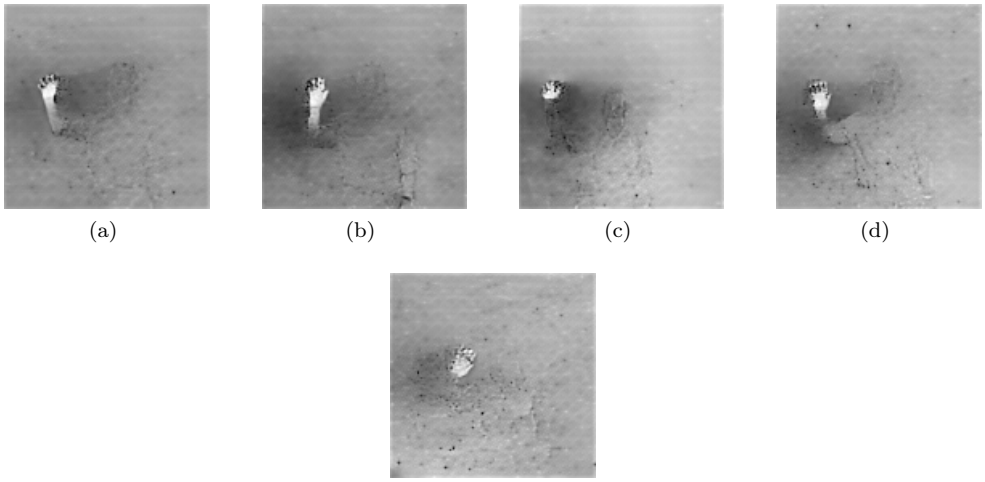
5.3.1 Intensity Reconstruction

The E2VID reconstruction network (as described in section 2.4) 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 TODO: Add image examples . This meant that 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 space 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.

TODO: Include image and explanations of NMNIST reconstructions

Figure 5.2 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. Since the reconstructions are consistent across lighting conditions, this also means that the reconstruction is more reliable, and the classification algorithm works better in adverse conditions in general TODO: get figures and images to prove this .



(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.

5.3.2 Classification Results

NMNIST Dataset

The confusion matrices for the classification of the NMNIST dataset can be seen in fig. 5.3. 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.

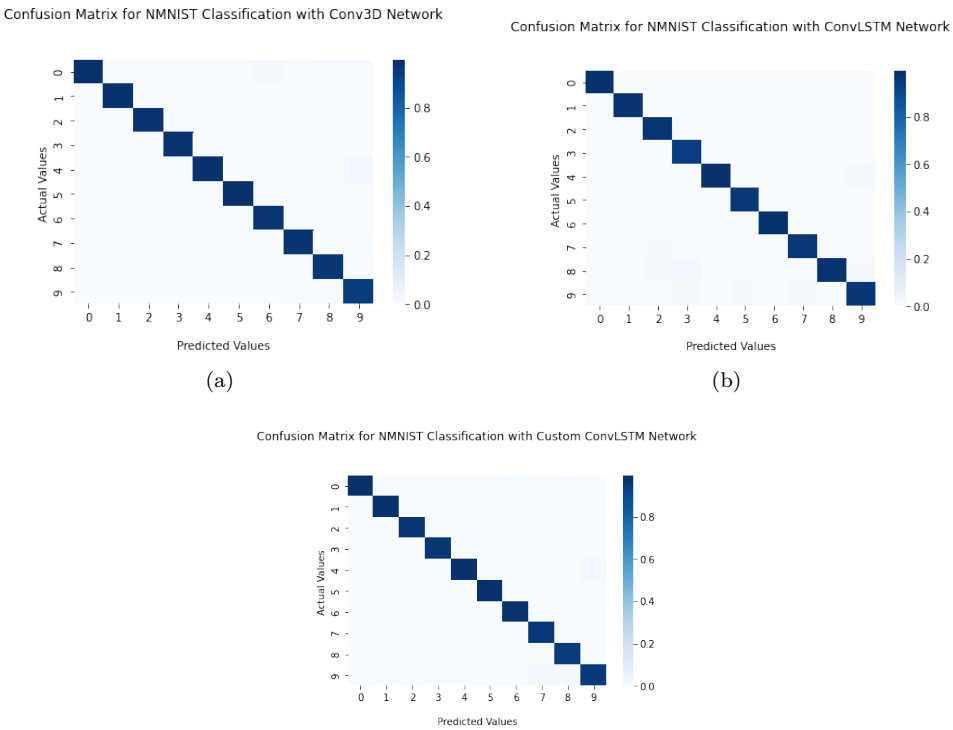
The drawback to the 3D convolutional network is that the number of trainiable parameters (as can be seen in section 3.6.1) is much higher than the other networks, and so the training and inference times were considerably higher than the other networks as well (as can be seen in table 5.5).

Table 5.1, table 5.2, and table 5.3 show the performance evaluation of each of the networks on the NMNIST dataset in more detail. TODO: Write more about the meaning of the tables.

TODO: Make more custom convolution LSTMs

DVS128 Gesture Dataset

The confusion matrices for the classification of the DVS128 Gesture dataset can be seen in fig. 5.4. The variance in performance between the networks is more apparent in this dataset than for the



(a) (b) (c)

Figure 5.3: Confusion matrices for NMNIST classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

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, as well as actions 6 and 7, were often misclassified as one another.

TODO: Make more custom convolution LSTMs

5.3.3 Network Comparisons

The full table of classification accuracies can be seen in table 5.4. 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 frame, it is less able to detect the temporal patterns prevalent in the gestures.

The training times for each of the networks on the datasets can be seen in table 5.5. It is evident that the networks with a higher capacity had higher training times. The 3D convolution network in particular had much longer training times than the other two networks, since it had many more parameters to optimise via back-propagation. It should be noted that for the networks in red the system 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.

	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 5.1: A table showing classification evaluation metrics of 3D convolutional network on 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 5.2: A table showing classification evaluation metrics of convolutional LSTM network on 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 5.3: A table showing classification evaluation metrics of custom convolutional LSTM network on NMNIST dataset.

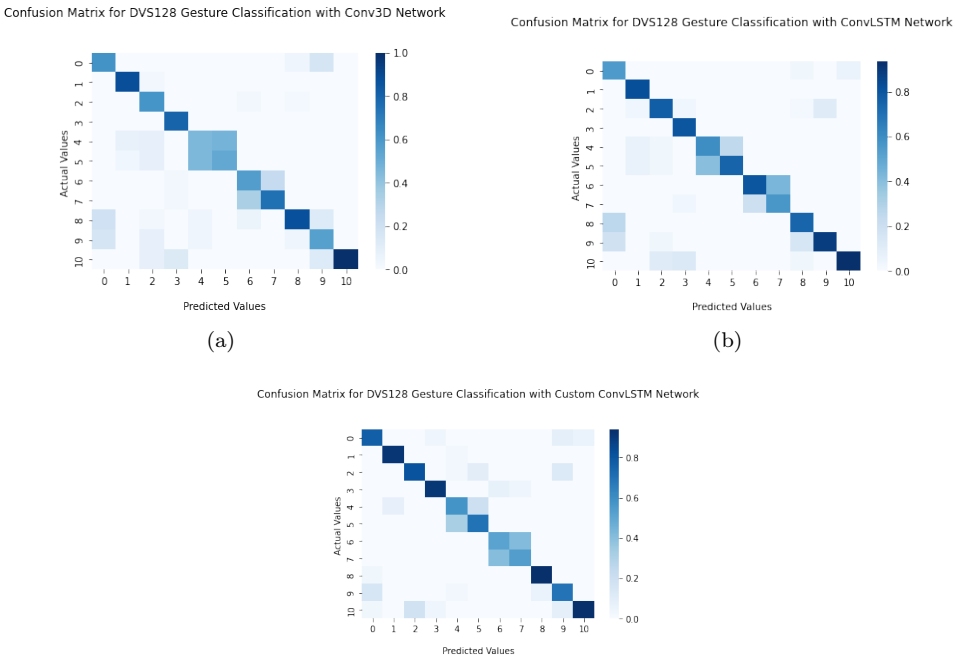


Figure 5.4: Confusion matrices for DVS128 Gesure classification with various networks; **(a)** conv3D, **(b)** convLSTM, **(c)** custom convLSTM.

Network	NMNIST	DVS128 Gesture
Conv3D Event Classifier	98.22%	69.44%
Conv2D LTSM Event Classifier	97.48%	71.53%
Custom LTSM Event Classifier	97.88%	76.39%
Conv2D LTSM Reconstruction Classifier	83.13%	19.26%

Table 5.4: A table showing classification accuracies of various models.

Network	NMNIST	DVS128 Gesture
Conv3D Event Classifier	1h43m3s	0h34m49s
Conv2D LTSM Event Classifier	1h3m9s	0h8m33s
Custom LTSM Event Classifier	1h12m10s	1h24m19s
Conv2D LTSM Reconstruction Classifier		

Table 5.5: A table showing training times various models.

Chapter 6

Evaluation

This chapter aims to follow on from the previous one which showed the results of each individual network. It serves as a high level overview of the performances of the end-to-end event classification pipeline and two-phase video reconstruction pipeline. There are also comparisons of each of the pipelines, allowing for the discerning their benefits and shortcomings.

TODO: Write out the following;

- for NMNIST the small size of images means that reconstructions do no vary much from the rudimentary intensity maps (could be used as an illustration of how reconstruction works).
- Similar to denoising network CNN more easily able to find patterns in data
- gestures are more complicated to characterise, since the networks for NMIST can easily learn all possible features. As well as this the image size for NMNIST is very small
- intensity maps can be of much higher framerate since events are taken asynchronously
- gesture intensity maps equivalent to 100fps but retains high frequency information without too much noise
- most gesture recognition datasets involve removing backgrounds and focussing on the hand, our network automatically only sees moving objects
- gesture recognition robust for even harsh lighting conditions

Chapter 7

Conclusion and Further Work

Appendix A

First Appendix

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] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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/

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

[27] 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

[28] 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/

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

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

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

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

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

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