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

May 23, 2022

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. A detailed implementation plan and project timeline for these objectives an be found in Chapter 3.

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.

Implementation Plan (Chapter 3)

This chapter has a list of planned objectives and their estimated completion date.

Evaluation Plan (Chapter 4)

This chapter presents a variety of ways in which the final project could be evaluated at each step. This allows for a comparison between existing solutions as well every iteration of solution created during the project.

Ethical, Legal and Safety Plan (Chapter 5)

This chapter presents any ethical, legal and safety considerations taken into account during the project.

Initial Implementation (Chapter 6)

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

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 Equation 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 Equation 2.1 into the Taylor expansion given in Equation 2.2. Equation 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 Equation 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.

Figure 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)**.

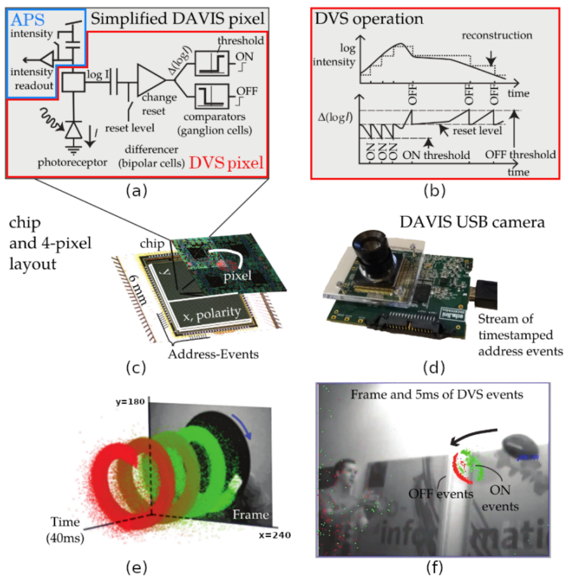


Figure 2.1: A summary of the functionality of the DAVIS event based camera[1].

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 Equation 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.*[9], 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[10], 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 Equation 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 Figure 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.

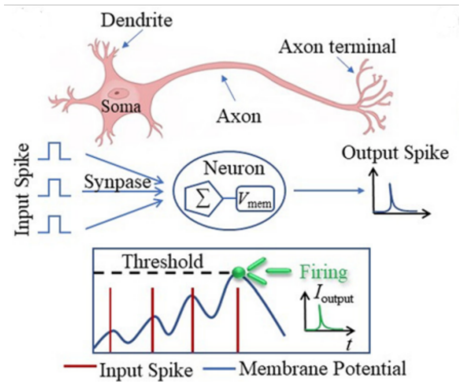


Figure 2.2: A diagram of structure and function of a biological neuron [2].

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.*[11] 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 Equation 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)[12]. Equation 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.

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

2.3.2 Localisation using Probabilistic Filters

Bayesian Inference

Unlike most other previous systems, probabilistic filters such as Bayesian filters are very much suited to work with data from event-based cameras. The reason for this is that it depends on iterative updates to the location probabilities using inputs, for which spiking inputs are ideal. Bayes's theorem can be derived from simple probabilistic rules[13]. We know $P(X|Y) = \frac{P(X,Y)}{P(Y)}$ and similarly $P(Y|X) = \frac{P(Y,X)}{P(X)}$. Therefore we can re-arrange both to give $P(X|Y)P(Y) = P(Y|X)P(X)$, since $P(X,Y) = P(Y,X)$. Then from there formula for Bayesian inference can be trivially obtained, as shown in Equation 2.6.

$$\begin{aligned} P(XZ) = P(Z|X)P(X) &= P(X|Z)P(Z) \\ P(X|Z) &= \frac{P(Z|X)P(X)}{P(Z)} \end{aligned} \tag{2.6}$$

In Equation 2.6, X is known as the prior (which is the assumed location of the camera) and Z is known as the posterior (which is the measurement taken by the sensor or camera). $P(Z|X)$ is known as the likelihood function, which indicates how likely it is to have received a particular reading given the assumed position.

Monte-Carlo Localisation

Now that we have the concept of Bayesian inference we can adapt it to create an efficient localisation algorithm. It includes initialising a number of particles that act as predictors of where in the map the camera is. We can now give the probability of a posterior camera position given a sensor reading.

The probability distribution $P(X|X)$ is a continuous function, and so updating each of the posteriors for every value of X is a computationally difficult problem. We can instead break it up into smaller bins to alleviate this issue. When generating these particles we can represent the probability distribution, albeit at a lower granularity. The benefit of this is that even though we cannot see the full distribution the peaks (i.e. the locations the camera is most likely to be in) are very well defined.

Now we can use the above simplification to carry out the following steps:

1. Randomly assign particle distribution across map.

2. Apply Bayes' law to measurement to update particle distribution.

   Bayes' rule can be simplified to be:

   $$w_{i_{x+1}} = P(z|x_{i_x}) \times x_{i_x}$$

   This can be done since the ignored multiplier on the right hand side will be normalised in the next step.

3. Normalise particle weights.

   Now the particles will have weights that no longer sum to 1, and so we need to normalise them again to follow the usual rules of probability:

   $$w_{i_{x+1}} = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

4. Re-sample particle distribution.

   We now need to create a new set of particles that all have the same weight ($\frac{1}{N}$), but whose spacial distribution reflects the new probability density.

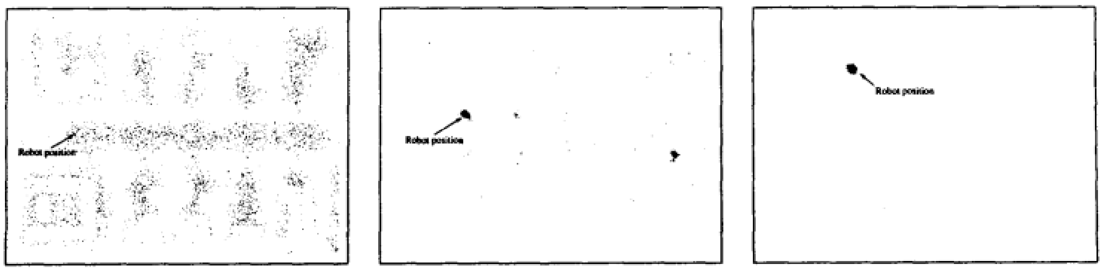


Figure 2.3: An example of Monte-Carlo localisation[3].

Figure 2.3 shows a typical example of the algorithm. The leftmost panel shows the random initialisation (or previous particle distribution), which then becomes the centrally shown distribution after one iteration. Since only one measurement is taken and the room is symmetrical, it is possible that it could be in one of two location (hence the two dense clusters). After one more reading in the next iteration, the algorithm is quickly able to narrow down the location of the robot. We can also see that any movement of the robot causes some noise to be added to the known robot location as we are estimating the location based on simple odometry using hardware such as wheel encoders to estimate the robots motion.

2.3.3 SLAM Algorithm

The Monte-Carlo localisation technique is useful for estimating a vehicles position (i.e, its location and orientation) given a model (or map) of the environment surrounding it. The next step is to carry out pose estimation of a robot while simultaneously generating a map of its surroundings (as shown in Figure 2.4). It is clear from the diagram that the equipment is not perfect and in an idealised environment. There is uncertainty in both the robot position (since there is uncertainty in the robots odometry), and there is also uncertainty in the measurements the robot takes.

There are two main branches of SLAM algorithms; filtering methods and smoothing methods. The former includes methods such as the Extended Kalman Filter (EKF) and particle filtering (similar to the Monte-Carlo filtering explained earlier). With these methods the state is estimated iteratively on the job as latest measurements are input into the system. The latter uses the set of complete measurements to estimate the full trajectory of a robot. Pose (or factor) graph optimisation is one such smoothing method that has become exceedingly popular in modern-day SLAM solutions.

The pose graph optimisation technique relies on memorising the robots relative position and the readings it took in that position. For example we can see a possible robot trajectory in Figure 2.4.

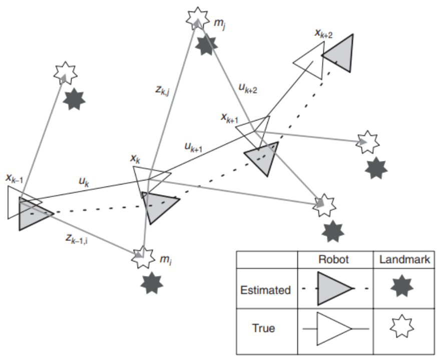


Figure 2.4: A diagram showing the fundamentals of the SLAM problem[4].

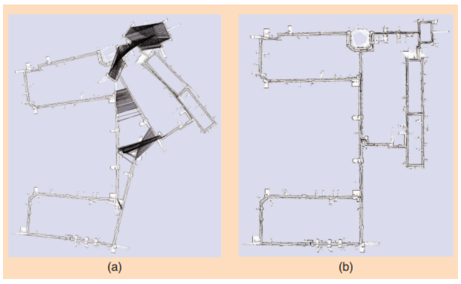


Figure 2.5: A diagram showing loop closure and optimisation steps of SLAM pose graph optimisation technique[5].

The robot and its new relative positions are all nodes of the graph and are connected by edges. Whenever a new node is created in the graph, a reading is also taken and stored. Then whenever a new node is visited it is compared with previously stored readings, and if there is a reading that is similar to a very high degree, 'loop closure' can take place. In essence this means that the robot is now visiting a location it has already visited, and so the nodes can be joined together. Then once each the loop has been established the graph has to be optimised so that each edge (and therefore node) is at its most likely position relative to the other edges. The way this is done is that each edge of the graph has a relative certainty associated with it, and this certainty dictates how flexible that particular edge is in the optimisation process. Once the loop has been closed the edges are moved such that the overall certainty of the graph is maximised. Once this is complete the readings can be stitched together to form a map of the environment, and the location of the robot within it is very well defined. This process can be seen clearly in Figure 2.5, where in **(a)** loop closure takes places, and **(b)** shows the map created after the subsequent optimisation phase. The maps are created using 'binary occupancy grids' or 'probabilistic grids'. The former simply stores binary 1's and 0's on whether a particular section of the grid is occupied or not. The latter adapts this by having probabilities of occupancy for each section rather than just binary values.

These SLAM algorithms and localisation algorithms in Section 2.3.2, however, are classical and have been mostly superseded by deep neural networks in most modern day applications. Furthermore, the algorithm in Section 2.3.2 is only applicable to localising the robot, whereas we want to be able to simultaneously map it's surroundings, for which SLAM was developed. Both these tasks have now been efficiently solved by neural networks for classical frame-based data, but there is still much ongoing research on how to do the same with spiking data.

2.4 Image 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'[14] 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 [6]. These methods are purported to allow the introduction of mainstream computer vision research to event cameras. Figure 2.6 shows an example of how converting spiking data to a video stream allows for use of classical computer vision algorithms.

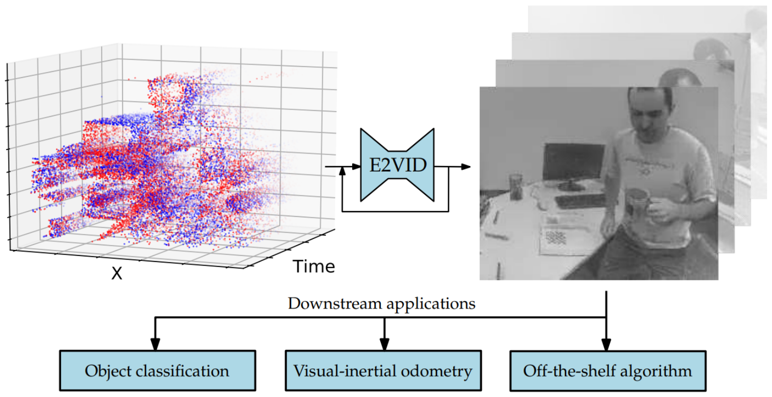


Figure 2.6: An illustration of the mapping of spiking data to video stream to apply off-the-shelf algorithms to[6].

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 2.7 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.

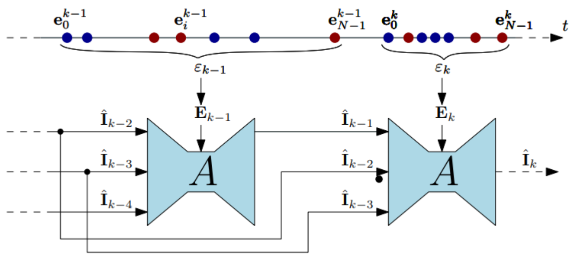


Figure 2.7: An overview of RNN used to generate video from sets of events[6].

2.4.1 Black-box 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 2.8 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).

In a paper by Arnon Amir *et al.*[7] 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 Figure 2.8 to act as

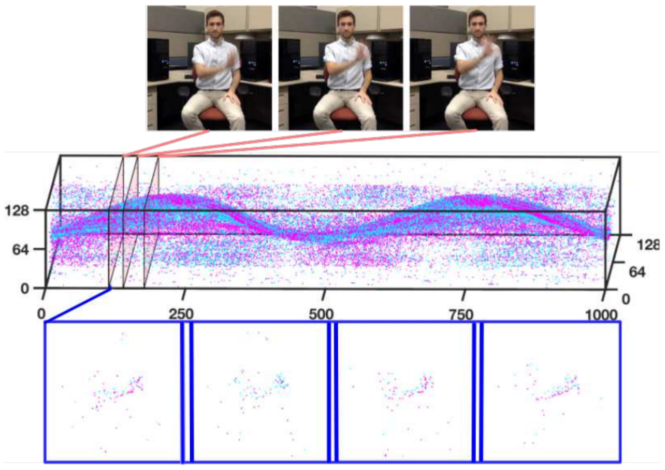


Figure 2.8: An illustration of a temporal filter that caches events fired from a camera[7].

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 Figure 2.9. 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.

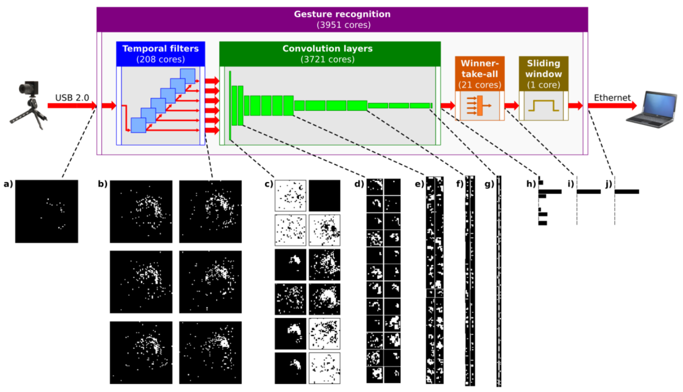


Figure 2.9: A network designed to perform gesture recognition using neuromorphic input by making use of cascading temporal filters[7].

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.*[15] moves towards building time surfaces from events to feed into a neural network, whereas Yin Bi *et al.*[16] propose using a non-uniform sampler to create a graph from events to then feed into a network of so-called graph convolution networks.

2.5 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.

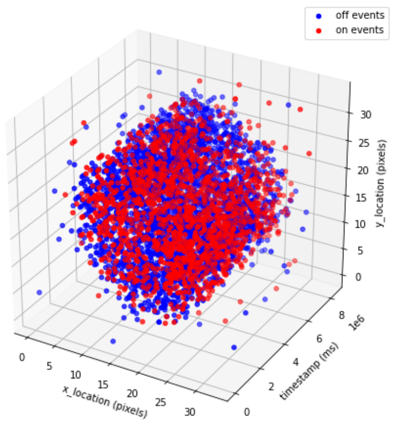
2.5.1 Neuromorphic-MNIST and Other Neuromorphic Datasets

This dataset is a spiking version of the original frame-based MNIST dataset [17][8]. 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.

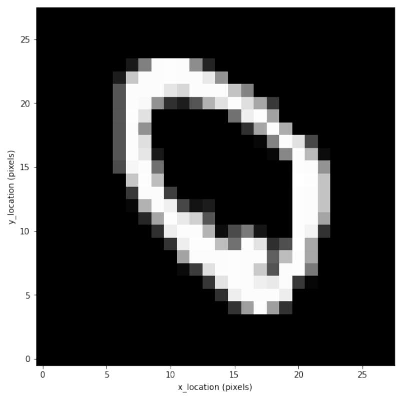
For each item in the dataset there is a binary file which has a list of events. Each event is characterised by a 40 bit unsigned integer. The integer gives the following information of a particular event:

- 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 Figure 2.10, 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 2.10: A visualisation of events from a single training sample from the NMNIST dataset.

More examples of neuromorphic datasets include:

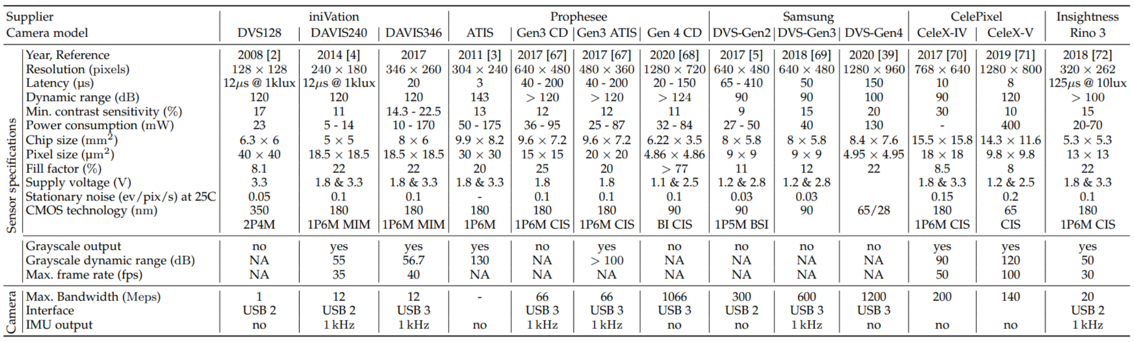
- **DVS128**

  This dataset is a set of 11 hand gestures from 29 subjects under 3 illumination conditions. It was created to help create a real-time gesture recognition system that utilises the low power capabilities of event-based cameras[18].

- **Heridelberg Spiking Datasets**

  The Spiking Heidelberg datasets for spiking neural networks[19] 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.

The above 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 Figure 2.11.



	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)	$12\mu s$ @ 1klux	$12\mu s$ @ 1klux	20	3	40 - 200	40 - 200	20 - 150	65 - 410	50	150	10	8	$125\mu s$ @ 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 2.11: A table listing widely available event-based cameras and their respective features[1].

2.5.2 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.*[10]. This avoids having to own an event-based camera to create data (as was done with the NMNIST dataset mentioned in Section 2.5.1), which is useful since the cameras are expensive and difficult to get a hold of in general.

Chapter 3

Implementation Plan

3.1 Timeline

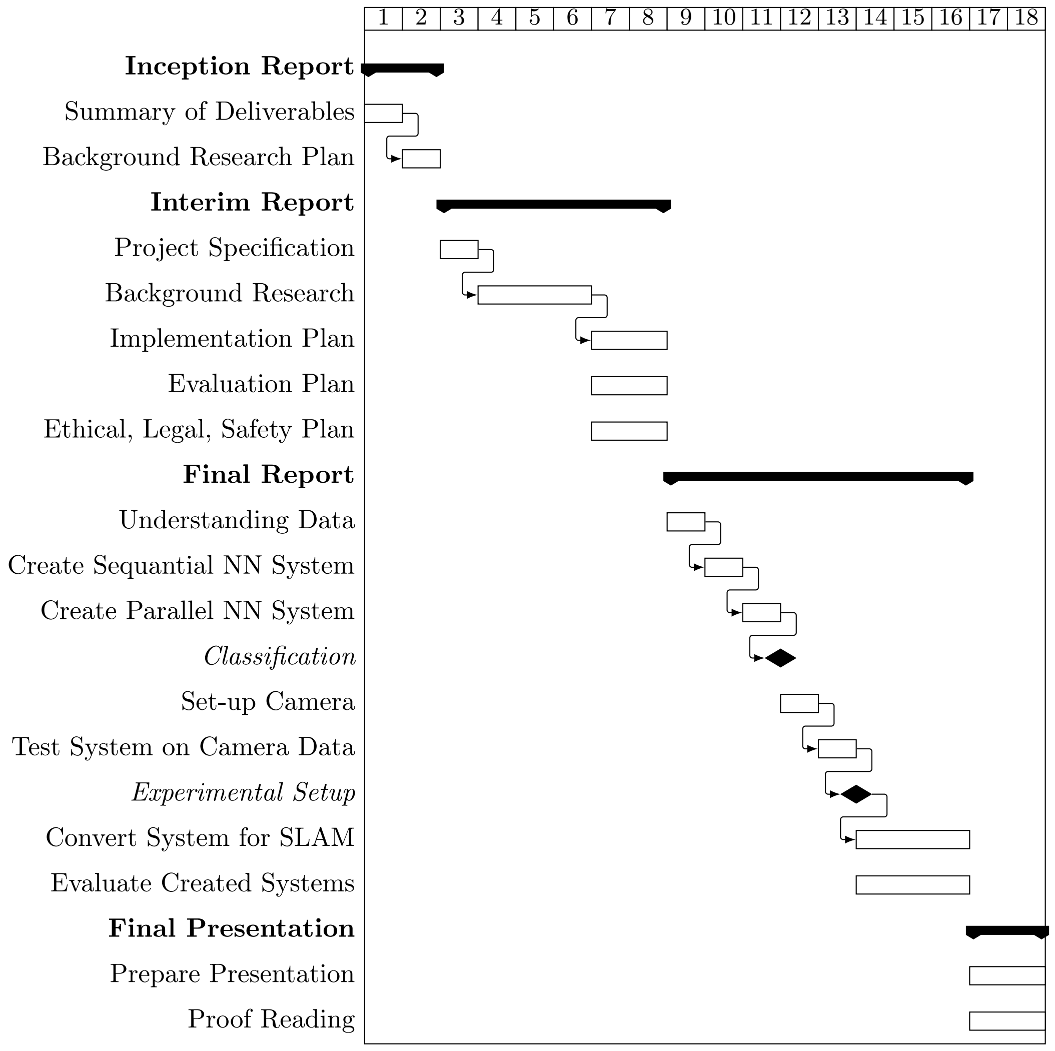


Figure 3.1: A gantt chart showing fortnightly progress. Main deliverables are written in **bold**, and milestones are written in *italics*.

3.2 Objectives

Name	Description	Timeline
Summary of Deliverables	Create an initial plan of deliverables to be present in final project. As well as this it is important to have some initial plans for fallbacks to ensure that a complete project can be achieved.	22/10/2021 → 5/11/2021
Background Research Plan	Create an initial list of relevant papers to kickstart the project.	5/11/2021 → 19/11/2021
Project Specification	Create a specification that precisely defines the goals of project as well as fallbacks for each case.	19/11/2021 → 3/12/2021
Background Research	Use initial list of papers to conduct research behind the topic of the project. This involves looking at previous works and their respective gaps in knowledge. This is in order to give a basis on which to begin implementation, as it should provide all necessary tools and knowledge to begin initial preparations. It is also useful so give an insight on the need for the project and what problems it aims to overcome.	3/12/2021 → 14/01/2022
Implementation Plan	Using analysis from background reading create a structured implementation plan outlining objectives and milestones, including a deadline for each.	14/01/2022 → 31/01/2022
Evaluation Plan	Create a structure for the evaluation of any implemented material, including detailed explanations for any formulae and significance of any values when compared to baselines.	14/01/2022 → 31/01/2022
Ethical, Legal, Safety Plan	Critically evalutate any ethical, legal or safety concerns that may arise as a result of this project and outline various ways in which to mitigate any possible issues.	14/01/2022 → 31/01/2022
Understanding Data	Load and evaluate the utility of various data sources and pre-process them ready for use in future NNs.	31/01/2022 → 25/02/2022
Create Sequential NN System	Create model that first reconstructs frame-based video from neuromorphic data to then pass into adaptations of existing networks that carry out classification on frame-based videos.	25/02/2022 → 11/03/2022
Create Parallel NN System	Create a single NN that takes neuromorphic data and directly creates the required output (classification or SLAM).	11/03/2022 → 25/03/2022
Set-up Camera	Create set-up to obtain real-world data from event-based camera. This also required to create a pipeline to process data to be used in created NNs.	25/03/2022 → 08/04/2022

Name	Description	Timeline
Test System on Camera Data	Since real world data may be less idealised than pre-existing datasets, it is important to test the functionality of the system with data obtained from camera. As well as this tests can be done under various more extreme circumstances to test robustness of system.	08/04/2022 → 22/04/2022
Convert System for SLAM	Adapt function of system and train for SLAM rather than classification.	22/04/2022 → 29/05/2022
Evaluate Created Systems	Use metrics listed in the evaluation plan to check the performance of each created system in order to compare them to each other as well as other existing solutions from other researchers.	22/04/2022 → 29/05/2022
Prepare Presentation	Create slides and content to present the final project results.	29/05/2022 → 06/06/2022
Proof Reading	Check through the report and presentation to remove errors and make any last-minute changes.	29/05/2022 → 06/06/2022

Table 3.1: A table of explanations of objectives given in Figure 3.1.

Chapter 4

Evaluation Plan

The evaluation plan is as given by the typical machine learning pipeline[20].

4.1 Dataset Preparations

4.1.1 Dataset Splitting

It is commonplace to split the shuffled dataset into three segments; training, validation and testing. The training data is what is used during each iteration of the back-propagation process. The validation data is what is unseen during this process and is instead used to give an estimation of a models performance while training hyper-parameters. Finally, the testing data is withheld until it is needed to compare different final implementations with each other. It is vital the the training and validation data are withheld while training since otherwise they would not serve as a simulation of unknown data to measure the performance of the system in an unbiased manner. Common splits for training, validation and testing datasets are 60%/20%/20% and 80%/10%/10% respectively.

4.1.2 Dataset Cross-validation

With smaller datasets the splitting of data may mean that there is too little left to train with. This problem can be alleviated by using cross-validation. This method entails dividing the dataset into a certain number of segments. Then in the first iteration of the learning process, the first segment is used as testing data while the rest is used as training and validation. Then in the next iteration the process can be repeated by using the second segment as the testing, and so on until each segment has been used as testing data. Finally we can use the average of the errors for each testing dataset as the 'global error estimate'. It can be noted that the same segmentation and iteration process can be used for the training and validation datasets.

4.2 Evaluation Metrics

For a classification task, when we obtain the results from the test dataset (as shown in Table 4.1) we can calculate a variety of evaluation metrics that give various insights on our final model.

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

Table 4.1: A table showing an example of results when inputting test data from NMNIST dataset[8] into the final model.

4.2.1 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 4.2 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 4.2.6). For each of the cells the number of matching records are stored to calculate each of the evaluation metrics. The table includes True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN).

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

Table 4.2: a table showing one particular confusion matrix for NMNIST dataset[8] for class 1 as the positive class.

4.2.2 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 $1 - accuracy$.

4.2.3 Precision

Precision is the proportion of positively predicted samples identified correctly.

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

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

4.2.4 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.

4.2.5 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.

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

4.2.6 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).

4.3 Baselines for Comparison

In order to measure the performance of the system against current solutions it is useful to have a list of baseline performances. This way it can be inferred if there is an improvement being made by any newly created systems. Examples of systems that can be used include the reconstruction algorithm posed by H. Rebecq *et al.*[6] and existing gesture recognition algorithms for the DVS128 dataset[18] like the one posed by Arnon Amir *et al.*[7] (See both in Section 2.3).

4.4 Additional Testing with Camera

If the model were to train using only the datasets available, there is a risk that the data would be unbalanced and the network is training on a very specific set of idealised readings. Testing with extraneous data generated by a camera under various different conditions would allow for evaluating the performance of the networks on data that is completely unseen and dissimilar.

Chapter 5

Ethical, Legal and Safety Plan

In general the project in unproblematic, and poses minimal safety risks other than the ones presented when working with computers for extended periods of time (e.g., RSI, eye-strain, back aches etc.). In terms of ethical and legal considerations, however, it is important to make sure to use the practical camera setup in a sound manner. When collecting data from others it is important to be mindful of privacy issues. Consent must be taken from any participants if ever the need arises to take videos of others, but for the vast majority of project only videos of myself will be taken. As well as this it is important to consider where the system may be used if it is every put into production. Especially in IoT applications, privacy is of utmost concern, luckily with spiking data, if an intermediate video reconstruction cannot be obtained from the system, very little information can be discerned from spiking data. Therefore it would be very difficult to identify any individual within neuromorphic data itself.

Chapter 6

Initial Implementation

6.1 End-to-end Classification System

6.1.1 Data Preprocessing

Time-relative Event Segmentation

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 6.1 shows a visualisation of intensity maps created from the NMNIST[8] 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[17] sample recorded by the ATIS camera (As shown in Figure 2.10 in Section 2.5).

Once a set of projections was made for each sample from the NMNIST dataset, it could be fed into a neural network in parallel. Ths could be done by simply flattening each intensity map and feeding all the cell values to a fully-connected dense layer in parallel, or the maps could be fed in as images to a convolutional layer. For the convolutional layer method a 3D tensor could be generated by stacking each of the intensity maps against each other and feeding them all directly into the first layer of the NN.

Neural Network Input Structure

There were many ways with which a neural network could accept an input to the system. The ones considered for this project where;

- Flattening all the intensity maps from segmented event stream into a vector to be input to a single layer of neurons.
- Passing a 3D tensor to an convolutional layer as input.

6.1.2 Pure Convolution Network

Convolutional neural networks have been shown to be much more effective when processing images that networks built solely with dense, fully-connected layers 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 input to the system is in fact a 3D tensor of multiple images (i.e. the intensity map video generated from the camera events) a typical convolutional network would not

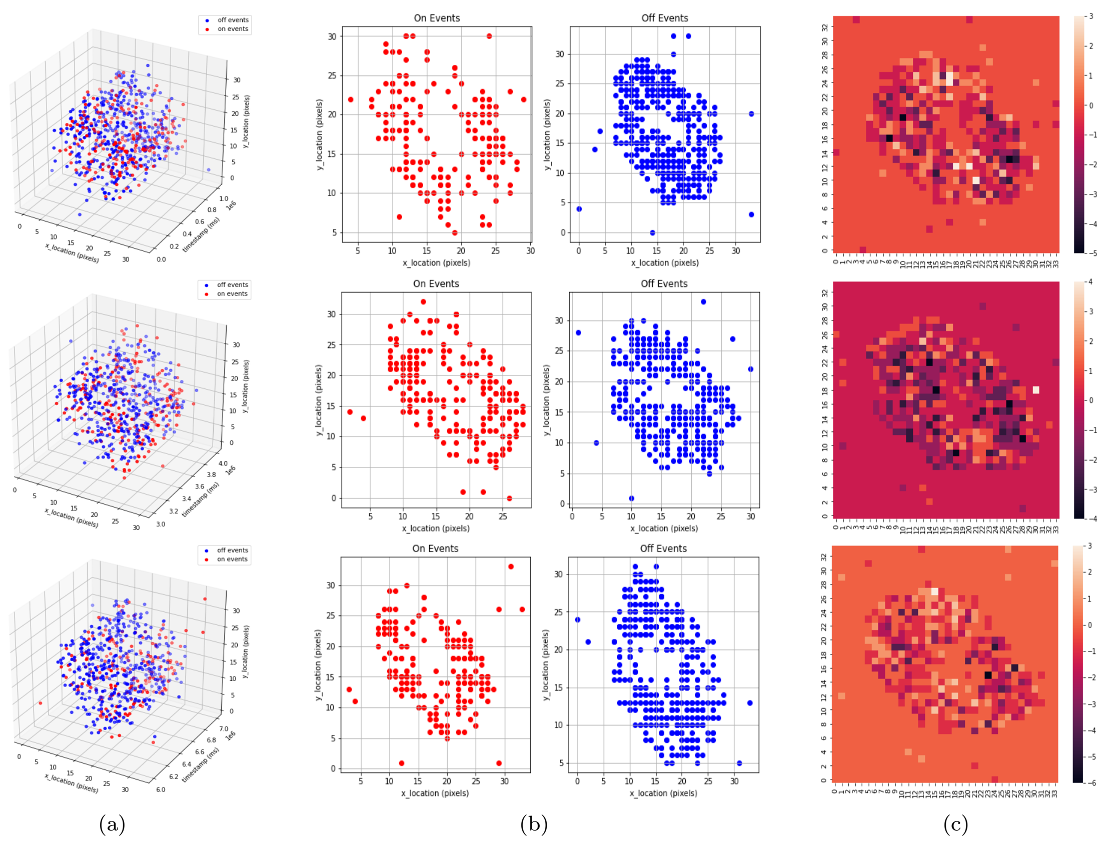


Figure 6.1: A visualisation of intensity maps created by segmenting events into bins of size 1×10^6 ms.

be 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 coudld be extended to more channels for each frame of the video, but this is not a scalable approach. Instead 3D convolutional layers were used [21]. Finally the outputs of the hidden layer were fed into a dense layer with 10 neurons to correctly 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 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 Equation 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. 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 Figure 6.2, where the performance on the training set steadily improves as the network progresses through epochs. Accuracy is one of the metrics defined in Chapter 4, 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.

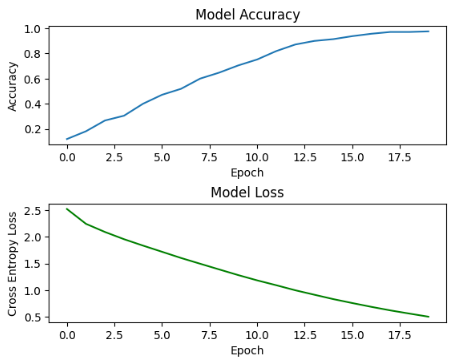


Figure 6.2: A figure showing classification accuracy and cross-entropy loss per epoch on training data for a typical network.

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

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 add image and reference here . 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.

Further testing was done with increasing kernel sizes, and with pooling layers added to the network. To find local patters filter kernels smaller than the image (32x32) are used. In earlier layers the no of filters is large and the filter size is small to capture details. The no of filters decreases and the filter size increases later in the network for higher-level patterns. Pooling filters the image after convolution layers to pick out important features. add final system architecture here .

6.1.3 LTSM

6.1.4 Convolutional LTSM

CITATION [22].

6.1.5 Spiking Neural Network

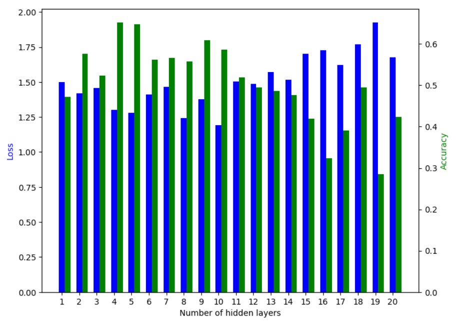
CITE NENGO etc.

Synaptic Smoothing

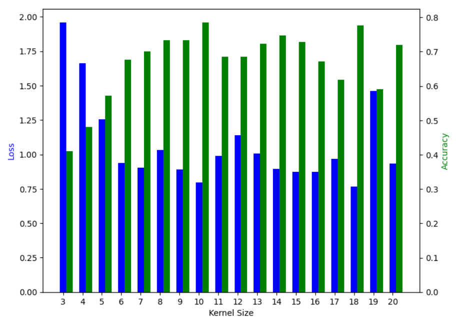
Firing Rates

Post-training Scaling

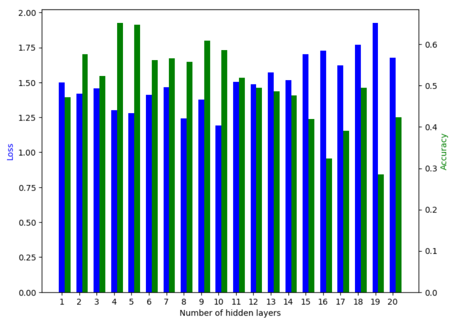
Regularizing During Training



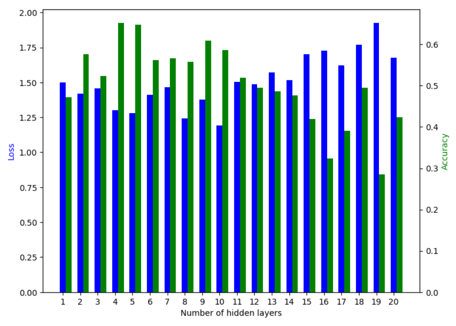
(a)



(b)



(c)



(d)

Figure 6.3: A visualisation of intensity maps created by segmenting events into bins of size 1×10^6 ms.

6.2 Video Reconstruction

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. It was interesting to note the performance differences between the previous networks and the described two-phase network.

6.2.1 E2VID

E2VID, as described in 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 CHECK)

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] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] T. C. Wallstrom, "The marginalization paradox and the formal bayes' law," in *AIP Conference Proceedings*, vol. 954, no. 1. American Institute of Physics, 2007, pp. 93–100.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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