Gaze Estimation with Event Cameras and Spiking Neural Networks

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2023

Andrew Lucas
Student ID – 9966879

Department of Computer Science

Table of Contents

Word count – 16, 919

Table of Figures

Declaration

This dissertation is all original work unless referenced clearly to the contrary.

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual Property Statement

i. **The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.**

ii. **Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.**

iii. **The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.**

iv. **Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy, in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library's regulations.**

1 Abstract

As Artificial Intelligence (AI) becomes commonplace in everyday life, the interest in deploying the technology in edge devices becomes more apparent. Although this would come with many benefits, such as lower latency and better security, it is challenging to use established models as they are often too computationally demanding for resource constrained devices. This paper investigates methods of sparse computation utilising energy efficient hardware in the form of event cameras and Spiking Neural Networks for efficient low latency computation. A real-world task, gaze estimation, is tackled with existing methods of processing event camera data, with results compared against a standard camera data baseline. The key issues with existing event data representations are identified before a new method of representing event data is presented that when used with a Convolutional Neural Network achieved a 7% higher accuracy at 6.05 vs 6.51 degrees error when compared against existing event representation methods whilst potentially requiring 89% less power to encode into a Spiking Neural Network. Compared to frames captured by a traditional camera the new method achieved an accuracy of just 1% worse at 6.05 vs 5.99 degrees error, with a sparser representation with the possibility for a 98% reduction in power required for encoding into a Spiking Neural Network. This paper starts by detailing the potential benefits of these technologies, before evaluating how they are currently used. Next the results of current methods are presented before the new method is described with detailed analysis of its performance. This paper shows promising results that could pave the way for these technologies being used more commonly for this and other similar tasks requiring low latency or power efficient vision, such as low latency obstacle detection in self-driving vehicles or low power object tracking in mechatronic picking systems.

2 Introduction

With the recent success of AI/machine learning models, there has been a substantial increase in interest [1] in deploying this technology in many areas of life across various domains. There is a significant challenge [2], however, in utilising these technologies in environments where computing hardware is limited or restricted by factors such as power consumption [3], for example in embedded devices such as mobile devices [4], mechatronic devices such as self-driving cars [5], or in robotics [6]. Attempts have been made to adjust AI models for these environments [7], but the most common approach to solve this issue is to utilise cloud computing and offload the computation to remote servers. This approach, however, introduces latency [8] and due to the inherently unreliable nature of network communication it is not well suited to real time systems, such as autonomous cars which are required to react to input and operate with extremely low delay and would be unable to support the loss of connection which is unavoidable with internet connections. This also creates other issues such as security concerns with data being uploaded and processed on a remote server [9]. Many influential recent advancements in AI do not account for hardware restrictions and the size of cutting-edge models has only increased over time. For example, recent Large Language Models (LLMs) have reached sizes of 70 billion parameters [10], widening the gap between the cutting edge and what can be run on many edge devices. There is also an ever-growing environmental concern with increasing power consumption of AI technology [11], which is again not commonly addressed in the world of AI research and will only continue to become increasing prevalent with the greater adoption of AI technology.

Artificial Neural Networks (ANNs) [12] are the backbone of most modern-day AI technologies [13]. This powerful computational model has led to a massive increase in the capability of AI [14]; however, they are notoriously power hungry, for example as far back as 2012, AlexNet a Convolutional Neural Network (CNN) a type of ANN required 725 million floating point operations for a single pass. Pruning [15] is a common approach to decrease the size of these models to allow for more efficient operation where neurons or connections in a network that have little effect on the output are removed. This, however, is done post training and does not reduce the power usage of training which often requires significant computational time and can only be done a limited amount before it leads to a decrease in the performance of models.

Another more recent approach is to make use of the sparsity seen in input data to reduce the amount of computation that must take place. This is, again, limited due to ANNs fundamental lack of support for sparse data, but 50% sparsity has been achieved by NVIDIAs Ampere architecture [16]. This achieved two times speed up in computation, with minimal effect on model performance when combined with pruning. Spiking Neural Networks (SNNs) are a different approach that can more optimally exploit the sparsity of data [17] to allow for more efficient computation and has been shown to be capable of achieving similar results to traditional ANNs in some tasks [18] [19]. These models, when run on specialised neuromorphic hardware, have shown great promise in achieving cutting edge performance with lower power usage [20]. For example, an SNN based on the ANN VGG-16 [21] is able to achieve an estimated 8.19 times reduction in power usage from $1.24e^7$ nJ to $1.58e^6$ nJ whilst maintaining comparable accuracy on the CIFAR-10 dataset [22]. This can be combined with methods to achieve more sparse representations of information to further the efficiency [22].

One method of sparsely capturing visual input, is using event cameras. Unlike traditional cameras, event cameras do not capture frames and instead report changes in illumination across pixels asynchronously. This means areas of images that haven't changed over time are not included in future data, allowing for the sparsity with only areas of interest, i.e. areas with motion being reported. In addition to this, event cameras can capture changes at much higher frequencies than traditional cameras with the Prophesee MetaVision Sensor [23] and the DAVIS346 from Inivation [24] capable of reporting changes at up to 10 000hz, making them of high interest for applications that require capturing fast motion or where low response time is critical.

An area where these technologies can potentially be of great use is gaze estimation. Where low power, low latency and high frequency processing would bring many benefits, particularly to on device systems. Gaze estimation is the process of taking visual input and predicting the point a target is looking at, often in the form of a pixel location on a screen. This has many applications, from user research [25] to mobile devices [26], or virtual reality [27] to even medical devices [28]. Since the eyes move at incredible speeds of up to 900 degrees per second [29], gaze estimation benefits from higher frequency data capture. Gaze estimation has been more recently used as a method of taking input from users allowing them to control devices with their eyes [30], such as in the Apple Vision Pro [31]. Currently, cameras used for gaze

estimation lag behind the frequency of common user input devices such as the mouse with only the highest quality hardware able to reach the mouse's typical 1000hz [32] [33]. This hardware, unfortunately, is often too bulky for use in many gaze estimation applications with more usable hardware being in the 25 - 60hz range [34]. The use of event cameras could decrease or eliminate this gap in input rate allowing for a better user experience in these applications. Additionally, these applications will also benefit from greater power efficiency as they are mainly used on battery powered, often lightweight devices, with limited hardware capabilities. Gaze estimation in recent times has been tackled, and has seen great success with deep learning based machine learning models [35] [36] [37]. This makes it an interesting challenge that could benefit from the use of event cameras to capture data alongside SNNs to process this input. To date, however, there is no established method for the best way to combine these technologies for gaze estimation.

The aim of this paper is to investigate methods of efficient AI for gaze estimation, looking into how sparse event camera data can be utilised and applied to this task and its suitability for use with spiking neural networks. Existing methods covered in a background of the state of the art of these technologies will be applied to determine whether these technologies can successfully be applied to gaze estimation. Results will be analysed to help produce a new method specifically for gaze estimation. The aim being, to achieve state of the art performance with increased efficiency, hopefully showing how these technologies can be exploited to achieve power efficient, low latency, high frequency computation, and inspire its deployment in other areas where hardware or power usage is restricted, or where low latency and high frequency would be beneficial.

3 Background

3.1 Gaze Estimation

Gaze estimation aims to predict the point a user is looking at given only visual input of the user. The output is usually an xy coordinate of a point on a screen [38] but is sometimes in the format of a gaze vector [39]. As datasets are most often created by getting users to follow points on screens, this is the most seen label in publicly available datasets [40] [41] [42]. Gaze vectors do not require a screen, so this is the only choice when using gaze estimation in scenarios that do not have users looking at a screen [43]. Gaze estimation accuracy is commonly reported as an angle [44] [45] [35] which allows the effect of the distance of the user from the screen to be removed from the accuracy, as demonstrated in Figure 3.1, allowing results to be easily compared across models and experiments. This is done by first calculating the absolute distance between the predicted and actual gaze point coordinates, before using that with the distance of the user from the screen to calculate the angle error.

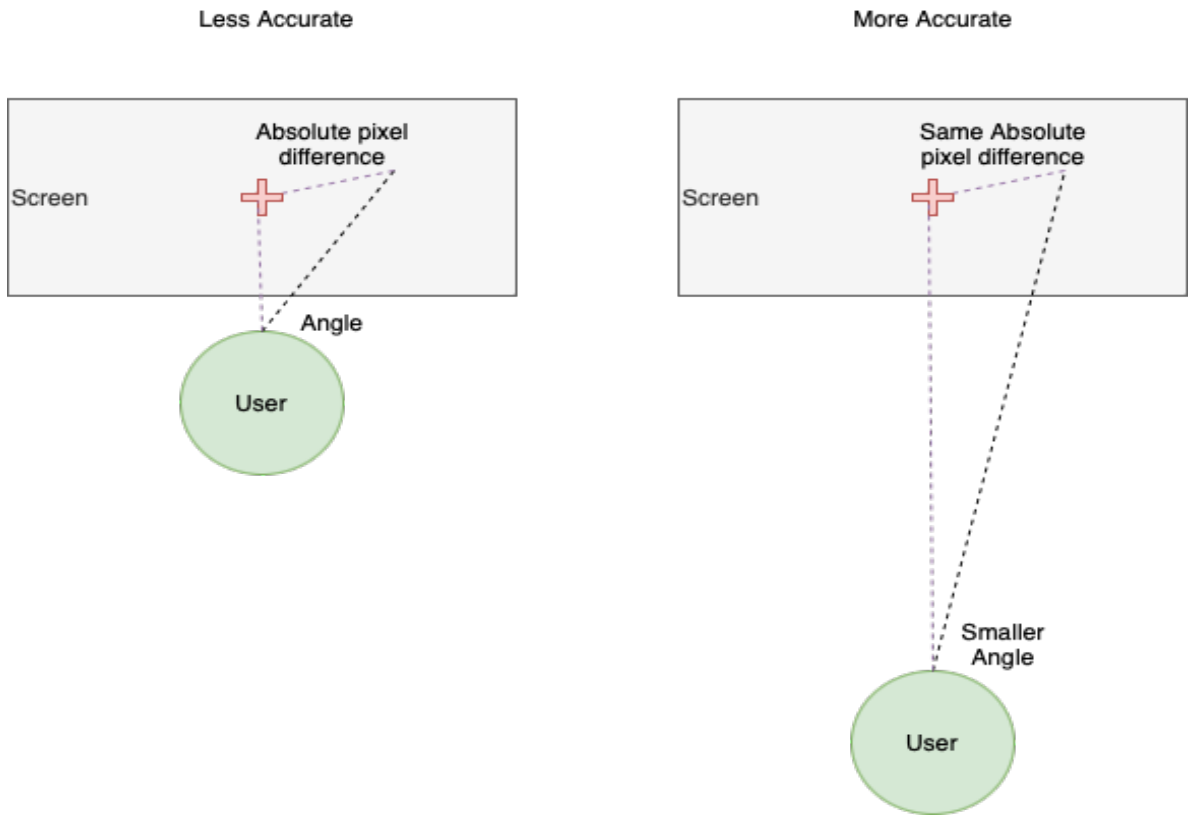

Figure 3.1 Diagram showing how reporting the absolute pixel difference as accuracy can be misleading due to different setups

Although gaze tracking and gaze estimation are often used interchangeably in the literature, there is sometimes a distinction between the two. Unlike gaze tracking where a user is tracked over time with an updating gaze point prediction [42], gaze estimation is done at a single point in time. Gaze estimation algorithms can be used for gaze tracking by repeatedly predicting a gaze point on sequentially captured data, where the accuracy is solely determined by the performance of individual gaze estimations. There are a variety of additional techniques that can increase accuracy using sequence processing models for gaze tracking such as Recurrent Neural Networks (RNNs) or Long Short Term Memory networks (LSTMs), though they often come with a large computational cost in terms of both processing time and memory requirements [46] which might not be worth the overhead, especially in scenarios where hardware is limited. As such, applying gaze estimation repeatedly for gaze tracking is still a valid approach in many cases, meaning the frequency of inputs and the latency of processing methods are still relevant to gaze estimation.

Gaze estimation can be tackled as either a regression or a classification task as seen in Figure 3.2. As a regression problem the model provides an exact x and y coordinate, or gaze vector given an input [47]. As a classification task a screen is split into sections and a model predicts the most likely area in which a user is looking [48]. This approach, however, limits possible accuracy as predictions are a lot less precise, so this is only used in situations where exact precision is not paramount.

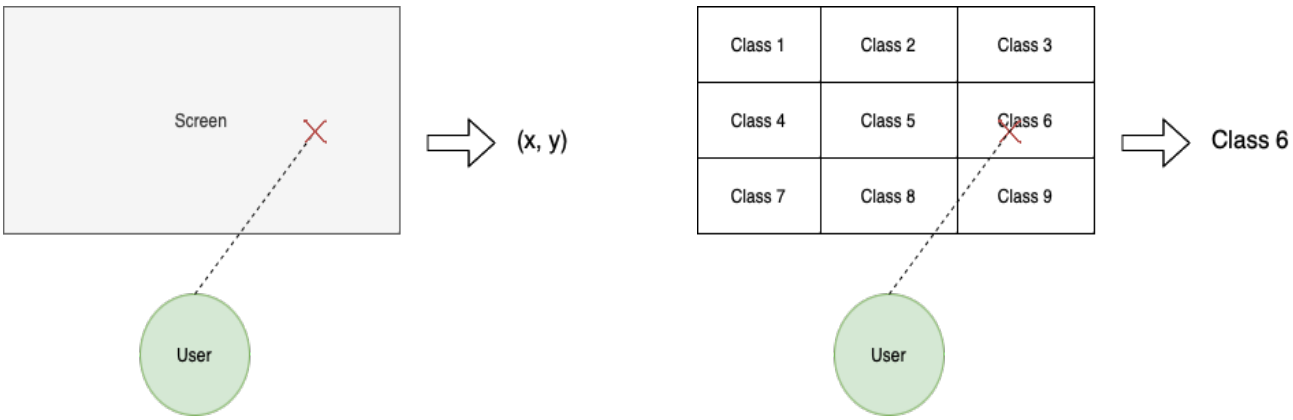

Figure 3.2 Diagram showing gaze estimation as a regression task (left) and a classification task (right)

One reason that gaze estimation is difficult is due to the speed that the eye can move. This can be up to 900 degrees per second [29], meaning large movements can be made in between captured inputs, e.g. between frames being captured. Additionally, blinks occur on average 12-18 times per minute [49], and can obscure the eye and make it impossible to identify the gaze direction. There are, however, favourable features seen on the eye that aid with the task. This includes the sharp change from white to black of the pupil, alongside easily identifiable features that are always present such as the eyelid. Additionally, some gaze estimation is done when a light is present that creates a detectable glint in the eye. As seen in the literature, and illustrated in Figure 3.3, these are the features commonly extracted in conventional non deep learning-based methods that are used to calculate the gaze point estimate [44] [45] [50] [51].

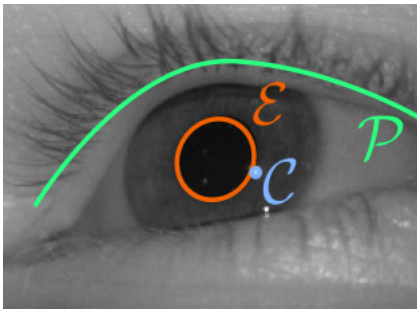

Figure 3.3 Image [42] showing the extractable features of the eye where ε = pupil, ρ = eyelid and C = glint

Due to the speed the eye can move, there is potential benefit in having higher frequency capture of data. This allows for more accurate tracking of small movements made by the eye allowing a clear picture to be made of the eyes' motion. A higher frequency will also allow for lower latency between eye movements and input into a processing device, allowing for quicker reactions to movements and a higher accuracy during these periods of movement. Unfortunately, computational cost increases linearly with the frequency of data processing highlighting the need for a more fundamentally efficient processing method.

There are two types of frequently seen gaze estimation datasets, as shown in Figure 3.4. The first is near eye gaze, where just the eyes are captured usually with a specially mounted camera which also commonly incorporates a light source which can aid in gaze estimation by creating a glint on the eye that provides another reference point which can act as a feature. The second is appearance based where a larger area is captured, for example the whole face. Many

appearance based algorithms work by first segmenting the eye and processing that and then using other features from other areas of the image to aid in predicting the exact direction of gaze [35] [40]. This means appearance based tasks could still benefit from advancements in near eye gaze technology by utilising similar approaches. Both types come with their own challenges and are representative of different scenarios where gaze estimation is applied, for example during mobile phone use for near eye gaze [38] and driver gaze estimation for appearance based [52].

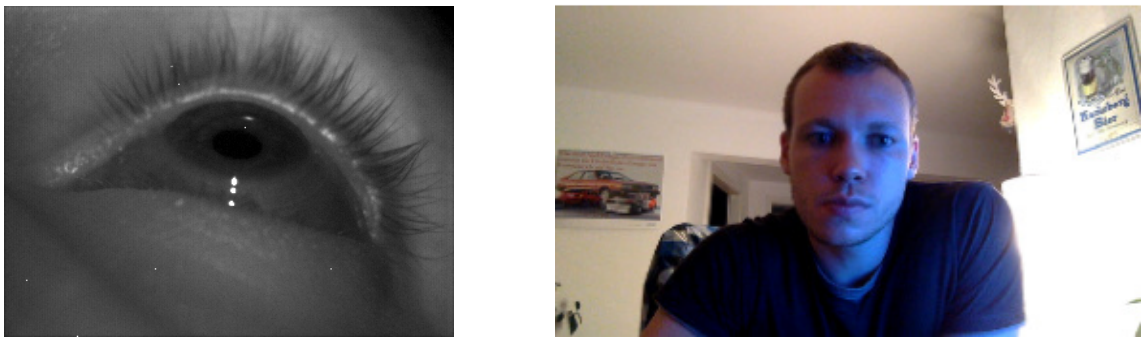

Figure 3.4 Example of near eye (left) [42] and appearance based (right) [40] gaze images

3.2 Existing Approaches to Gaze Estimation

Gaze estimation predates the dominance of deep learning in image processing tasks. As such, more traditional approaches exist that have continued to be developed in recent years. Referred to as model based approaches [44] [45] [50] [51], these algorithms use hand crafted features, most often the pupil, the eyelid, and a glint if one exists, as seen in Figure 3.3 which are extracted and passed into models that map the locations of the features to a gaze point. Regression can be used as a model to map the features' locations to a prediction [44]. Although these approaches have shown successful results, they often rely on a calibration stage where images are taken with the user looking at a known point so the regression model can be calibrated. This makes it unsuitable for many applications but provides a good baseline for target accuracy.

To combat this weakness, deep learning has been deployed to great success in achieving accurate gaze estimation without the need for calibration. Deep learning approaches seen in literature almost exclusively utilise Convolutional Neural Networks (CNNS) [47] [38] [53]

[54]. Although this has been used to great effect, the results don't quite match that of calibrated traditional approaches, with deep learning based approaches achieving around 4 degrees error at best on the most commonly used public gaze estimation datasets [55] [56] compared to as low as 3.3 degrees seen in calibrated model based methods [50]. As deep learning approaches generalise better, they can avoid the need for calibration, and as a result, it is much more useful in many applications. There is interest in using different kinds of neural network models, such as capsule networks [57], to tackle this problem. This is due to CNNs often losing spatial information during the convolutional layers because of the pooling that takes place [58], and even though neural networks are a black box, it can be assumed they follow a similar approach to traditional algorithms by first locating key features of the eye before mapping their positions in relation to each other to a gaze point. As such, this makes the first step an object detection problem which requires high spatial precision, meaning CNNs might not be the optimal choice. As they are usually so big, they still achieve good accuracy. There is a possibility that exploring other architectures that are better at handling spatial information could lead to higher efficiency. As an example, Capsule Networks can perform better with spatial tasks and have been used to achieve accuracy as low as 4.06 degrees [56] on the MPIIFaceGaze dataset matching the performance of the best CNN model on the same task. Capsule Networks in comparison to CNNs are built in a way that they do not only detect a feature but information about it such as its position and orientation in relation to other features. However, these models are much less established and there are few example applications to gaze estimation, so results are hard to verify with confidence.

Finally, approaches which tackle gaze estimation as a sequence processing task, whereby eyes are tracked over periods of time, using multiple frames have shown great results "achieving a significant improvement of 14% over the state of the art" [59], attaining an accuracy of 6.1 degrees on a harder dataset the established MPIIGaze algorithm [40] was only able to achieve an accuracy of 7.3 degrees. These approaches primarily use models such as RNNs, LSTMs, or (more recently) transformers [60]. Although these approaches can achieve good results, these sequence processing models come with great computational cost. As such, they are not appropriate for the main task of this paper, to investigate efficient approaches to gaze estimation for devices with restricted hardware. Additionally, the time needed to properly train and evaluate these models is outside the capability of this research paper. These methods will be

kept in mind as advancements in efficiently processing sequence data, such as further developments including transformers, could make this more practical soon.

3.3 Efficient Deep Learning

One method for adjusting deep learning models for hardware restricted environments is pruning. This technique involves selectively removing weights, neurons, or even entire layers from neural networks, as depicted in Figure 3.5, thereby reducing the model's processing or memory requirements. There are many algorithms that take different approaches to pruning [61], however over or under pruning is a common issue in many of them. Pruning must be carefully managed to avoid compromising a model's accuracy. Similar to the standard training phase, pruning employs a test set against which the model is evaluated as the algorithm systematically eliminates weights or neurons. The goal is to ensure that test accuracy does not fall below a specified threshold. An example of a very simple algorithm for pruning would be to list all weights by their value and start removing the smallest ones. Pruning relies on models being overparameterized for a task to meaningfully reduce model size [62]. Luckily this is often the case as shown by examples such as AlexNet being compressed to 35 times smaller than its original size [63] with only negligible effect on performance on the ImageNet dataset [64]. Due to this, you could theoretically get the exact same resultant model size by selecting a more efficient model structure, highlighting that pruning fundamentally supplements poor model selections [65].

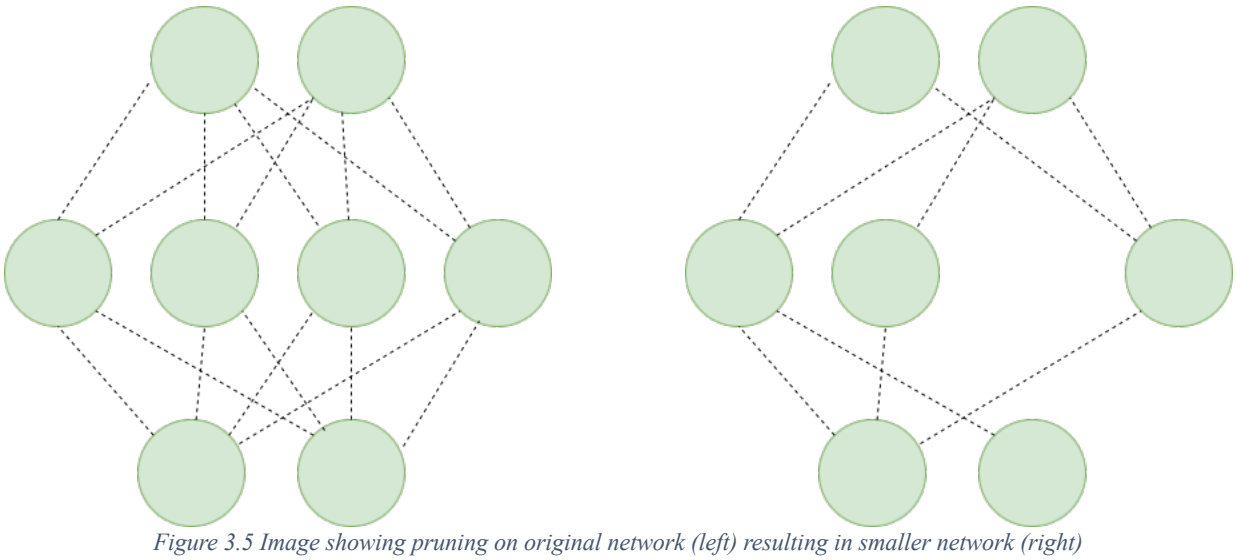

Figure 3.5 Image showing pruning on original network (left) resulting in smaller network (right)

Quantization is another approach used for lower computation and memory footprint. This involves reducing the resolution of weights by reducing the number of bits used to store them and rounding their original values to the nearest possible representable value in the reduced resolution [66], for example by converting from 32-bit integers to 8-bit integers. This not only reduces the memory required to store all the weights, but also makes calculations involving the weights computationally more efficient. This method is often applied with pruning and can provide good results [67], achieving a 37x times reduction in model size on ResNet-18 with only a 2.73% drop in accuracy [68]. But it again has limited applicability before performance is affected too much.

The NVIDIA Ampere architecture introduced methods of applying both pruning and quantisation combined with exploiting sparsity in data and claims to achieve up to 50% reduction in model size with negligible effect on performance (~ <1%) [16]. This method [69] works by first training a fully dense model and then retraining another model but with a 2:4 sparsity ratio in the input, which requires 2 in every 4 values to be 0 which are then dropped or just not processed. If this sparsity isn't naturally occurring in the data, it is quantised to achieve this before the model is trained. Then pruning is used to reduce the model size. Although this has promising results the structured nature of the required sparsity makes it unsuitable for most image processing tasks as images will usually have areas of interest grouped together and any possible sparsity grouped outside the areas of interest. Similar approaches may also fail because of the structure of images as it is hard to enforce sparsity on a traditional camera image. For example, for quantising to enforce sparsity you could convert images to binary images. It is then easy to see how this works for synthetic data, e.g., MNIST as seen in Figure 3.6, where areas of interest have sharp contrast to their background, but in real images of busy scenes there will higher frequency changes which might result in less meaningful structure in the binary image as too much information is lost, resulting in a low quality image that makes it hard to distinguish key features, an example being shown in Figure 3.7.

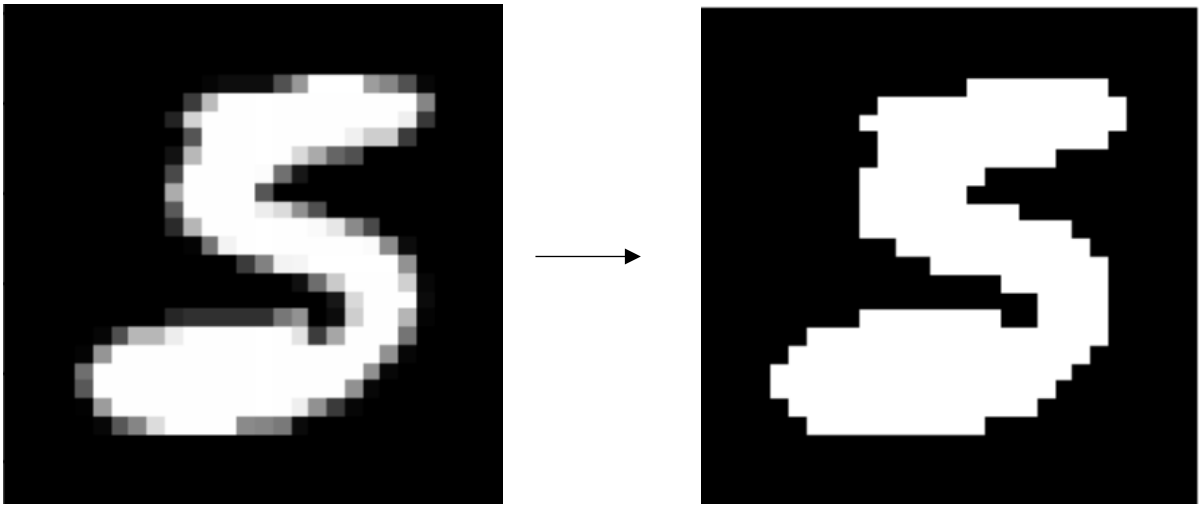

Figure 3.6 Example of MNIST digit getting converted to a binary image without losing structure

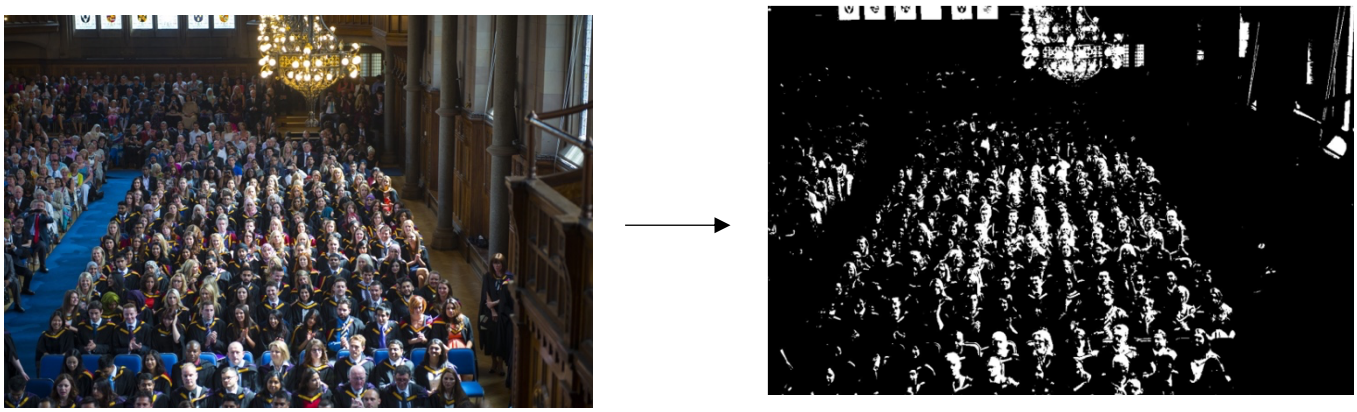

Figure 3.7 Example of a crowd of people getting converted to a binary image, resulting in a low-quality image

There are other methods of interest that don't directly influence model size but can increase the chance of training a smaller model that achieves required performance. These mostly revolve around identifying the ideal model structure and parameters. There is focus on algorithms to find the optimal hyperparameters for models which will maximise the performance for a given number of trainable parameters, which will allow for models to reach acceptable performance with fewer parameters, such as genetic algorithms [70] and Bayesian optimisation [71]. Another method, called knowledge distillation [72], first finds a large model that performs well and then creates a smaller model that doesn't learn directly but instead is trained to mimic the larger model, which for example, has been used to achieve a 5 times reduction in model size without any sacrifice to performance [73]. There are also approaches that are mainly targeted

at decreasing power usage during training by improving the speed of convergence or increasing the chance of finding a global extremum as to avoid needing to re-run training multiple times. An example is self-supervised pre training [74] where layers are pre-trained on unlabelled data by going through each layer and training it like an autoencoder to learn features before fine tuning the full model on labelled data. This has been shown to successfully achieve faster more consistent convergence to optimal solutions [75]. Though the primary purpose of this method is to allow models to be mostly trained on unlabelled data, which is easy to come by, before being trained on a more limited set of labelled data, or just to speed up convergence, and it isn't designed to directly influence model size, again it can increase the chance of successfully training a smaller model to successfully complete a relevant task [76].

3.4 Spiking Neural Networks

Spiking Neural Networks (SNNs) are neural network models that, unlike standard ANNs, make use of 'spiking' neurons. Similar to ANNs, SNNs are also inspired by the brain but take this concept further to attempt to better mimic the behaviour of the brain [77] by using the timing of binary spikes between neurons as a method of communication rather than continuous values, adding a temporal dimension to the network. In these models, neurons only fire when their stored value, called the membrane potential, reaches a specific value called the membrane threshold. Unlike in ANNs, output from neurons is not a continuous value calculated by passing its input through an activation function, such as a sigmoid or RELU (Rectified Linear Unit), it is a binary value with 0 representing no spike and 1 representing a spike [78]. Neurons communicate through these spikes in various ways including using the timing, called latency encoding, or frequency, called rate encoding. Like ANNs, inputs to neurons are multiplied by a weight. As the output from neurons is a binary value, however, the input to the next level neurons can be calculated as a sum of weights from neurons that have spiked, rather than a product of all previously connected neurons outputs and their corresponding weights [79]. This is one property of SNNs that can be used to optimise performance as, when a neuron spikes, the relevant weights can be loaded from memory and directly added without the need for any additional calculations.

SNNs process data in timesteps, during which individual neurons can send signals asynchronously and are processed by the end of the timestep irrespective of order [80]. This allows the use of parallel computing to greatly increase the throughput and reduce latency, which is not the case for normal ANNs that must process each layer in order. As nothing is done for a neuron when it doesn't fire, neuromorphic hardware is able to utilise a very low power 'sleep' mode during this period as another way to increase power efficiency [81]. A leak value is sometimes used to decrease the membrane potential between each timestep. Although in traditional SNN approaches only the weights between neurons are trained, in some approaches the membrane threshold and leak, also referred to as decay, are trained which has been used to great effect [82] [83], allowing models to be trained that require 6-18 times less energy with similar accuracy when compared to SNN models that do not use this approach. Increasing the number of timesteps has been shown to be beneficial for accuracy in some tests [84], but comparable accuracy has been achieved in the field with fewer timesteps [85], including as low as 1 timestep [83]. This is beneficial as it decreases the latency between the start and end of processing a single input [83] and increases power efficiency [86]. Training a single timestep SNN directly has not been done with acceptable accuracy, however this has been achieved by first training a higher timestep SNN and then iteratively going through steps to optimise and reduce the number of timesteps required by adjusting the membrane threshold and leak to account for fewer timesteps [83]. This has shown the possibility to create an SNN that achieves 25-33 times higher power efficiency with 5-2500 times lower latency whilst only sacrificing approximately 1% accuracy compared to higher timestep SNNs on a CIFAR-10 image recognition task [83].

A common model used for SNNs is the leaky integrate and fire model. This model includes a decay multiplier that decreases a neuron's membrane potential between each timestep and a reset term that subtracts the threshold membrane potential from the current membrane potential after a neuron fires. This is defined as:

$$U[t] = \beta U[t-1] + WX[t] - S_{out}[t-1]\theta$$

$$S_{out}[t] = \begin{cases} 1 & if\ U[t] > \theta \\ 0 & otherwise \end{cases}$$

[77]

Where *U[t]* is the membrane potential at timestep t. β is the decay multiplier that reduces the membrane potential each timestep. *WX[t]* is the new input to the neuron calculated as the sum of all the connected spiking neurons corresponding weights. $S_{out}[t-1]$ is the output of the neuron at the previous timestep, with 1 being a spike that will result in this term equalling the threshold which will be subtracted from the current potential, otherwise it will be 0 meaning the neuron didn't spike and this term will have no effect. θ is the threshold set for this neuron for which its membrane potential must exceed before firing [78].

There are many decisions that must be made when designing a SNN that provide varying results. One such decision is the type of input and output encoding/decoding to use. Common options include rate encoding, where the frequency of spikes is used to encode/decode data, and latency encoding, where the timing of spikes is used to encode/decode data [78]. To provide an example, a SNN used for classification rate encoding can be used by taking the output with the most spikes as the predicted class, and latency encoding can be used by taking the class that spikes first or last as the predicted class, as shown in Figure 3.8.

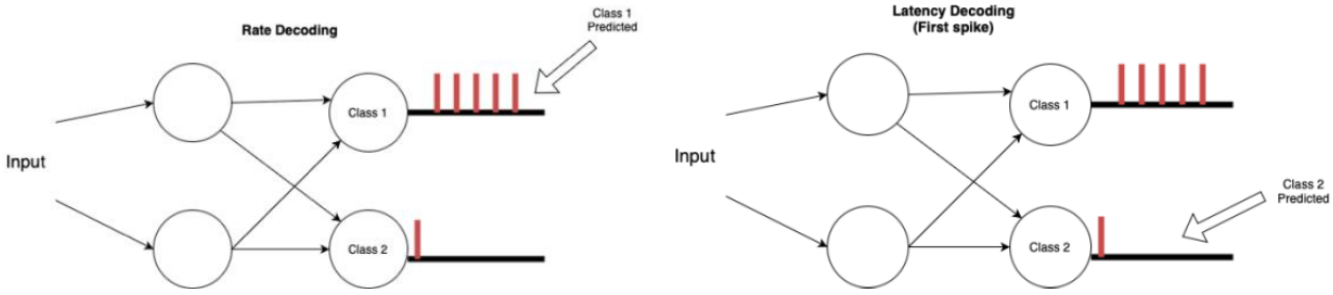

Figure 3.8 Diagram showing how rate and frequency decoding can be used for classification

Additionally, variations of these coding schemes exist, including the commonly used Time-to-First-Spike (TTFS) which uses the principle that more important values are passed first by having spikes at each timestep representing an exponentially decaying value that decreases over timesteps [87]. The precision of this is decided by the number of timesteps, with values having to be quantized to approximate values in the case of less timesteps being used. This scheme has the advantage that more impactful data is passed into the model sooner allowing

for the possibility of quicker response times [88] alongside having more precision in values whilst still only using 1 spike to encode them. An alternate method used for output when highly precise values are required, for example in a regression task such as gaze estimation, is to directly read the membrane potential on the output neurons after the last timestep as the output of the model [89].

Another choice to be made when using SNNs is the selection of the learning algorithm. Due to spike signals being binary step functions, which have an undefined gradient at the activation point and a 0 gradient elsewhere, they are non-differentiable [90]; backpropagation cannot be used without additional methods being employed. Example solutions to this include shadow training, where an ANN is trained before being converted to an SNN [78], backpropagation through time (BBPT) combined with surrogate gradients [91] [92], where, as with RNNs, the model is unravelled to allow back propagation to be applied whilst substituting in sigmoid or fast sigmoid functions for the spikes when calculating gradients [78].

The power used by an SNN is positively corelated to the number of spikes that are processed, with the major contributing factor being the amount of memory reads that must take place [86]. It follows that to decrease the power usage, the number of spikes should be decreased as much as possible whilst maintaining good accuracy. Some approaches to SNNs do not account for this and use spike trains that use lots of spikes using a Poisson spike generation process, [92] however there are methods that seek to minimise spikes, one of which being learning algorithms that aim to train SNNs in a manner that minimises the required spikes by including the number of spikes as a penalty term in the loss function [93]. As the success of reducing power usage is hard to track without the specialised neuromorphic hardware see the full benefits, this paper focuses on minimising the spikes in the input layer which can be easily determined by calculating the number of spikes needed to encode the data. This in turn, will serve to decrease the number of spikes a network must process even if just at the input layer.

3.5 Event Cameras

Event cameras are a type of camera that report changes in log luminance asynchronously rather than capturing complete frames as seen in traditional cameras. Events are triggered according to the following formula for each pixel:

$$\pm P = \log(x, T) - \log(x, T - \Delta T)$$

[94]

Where *P* is the threshold that must be met to trigger a corresponding negative or positive polarity event. log*(x, T)* is the log luminance at time *T* and log*(x, T* - ΔT*)* is the log luminance at the time when an event was last triggered. ΔT is the time since the last event.

These cameras can capture changes at extremely high frequencies providing very low latency at as low as 1 microsecond [94], have high dynamic range at 140Db vs the typical 60Db seen in traditional cameras [94], use very little power averaging 1mW vs 1W of traditional cameras [94], have no motion blur, work in low light environments, and often have a small profile making them extremely attractive for use in embedded or edge devices [23] [24]. Although they are of particular interest in low power systems, they can also provide benefits over traditional cameras in other systems where these properties can be beneficial. There are drawbacks to event cameras, however, mainly due to relying on illumination changes, since no pixels will be triggered in a static scene, surfaces often will not trigger events for the same reason meaning only edges are captured. Edge detection is often the first step in image processing algorithms, however, so they can still provide enough information for these algorithms, potentially simplifying the process [95] [96]. Furthermore, they have noise in the form of spurious events triggering which can often occur in a burst in the same location over a short period.

Additionally, despite the many potential benefits of using these cameras there is still a lack of appropriate algorithms available for processing this data effectively. Event camaras usually report data in the address-event representation (AER) format where events are reported as a tuple containing the timestamp, the polarity of the event, e.g. 1 being positive and 0 being

negative, and the xy coordinate in which the event occurred [97]. In practice this creates a long list of these events that have no structure such as that of a traditional image frame, as illustrated in Figure 3.9. As such, normal image processing methods cannot be used on this data directly. Most approaches to date start off by converting this data into a format like image frames before using existing image processing algorithms [98] [99] [100] [101].

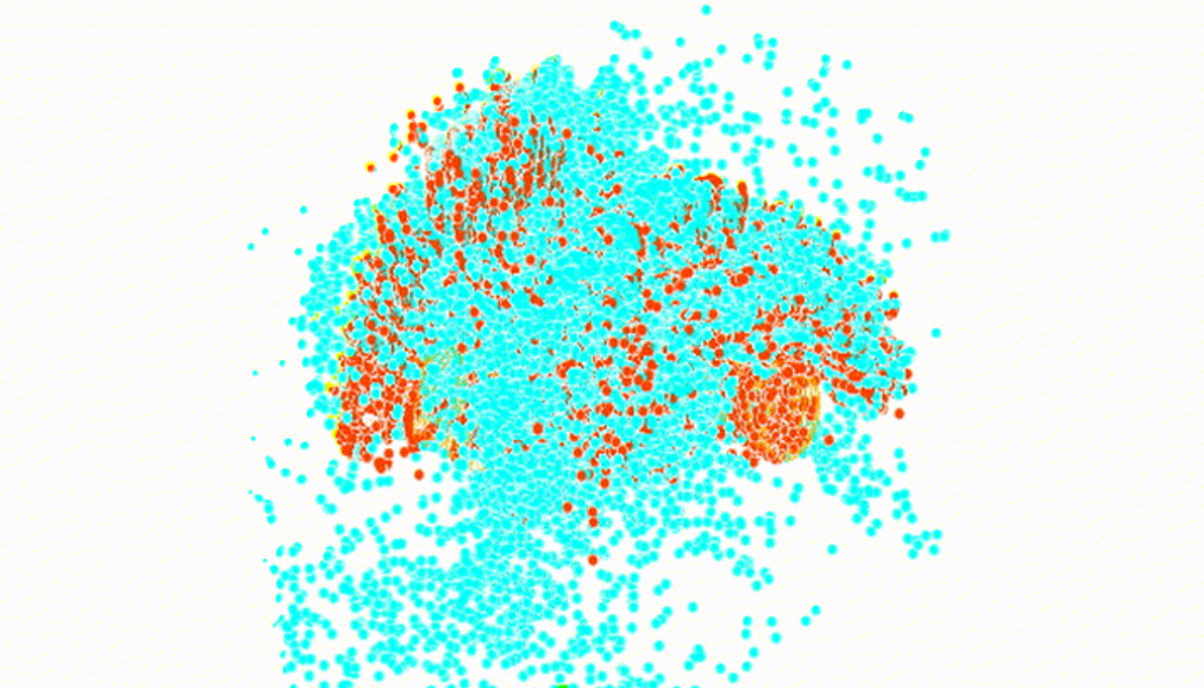

Figure 3.9 Visualisation of raw event data [42] where the x and y axis are the coordinates of the event, and the z axis is time. Blue dots show positive polarity events and orange negative

3.6 Event Camera Data Representation

There are many techniques that can be used to build meaningful representations from event data that can be used in a similar way to traditional camera frames for image processing. In this section, the most common of these will be described with a focus on their suitability for gaze estimation.

3.6.1 Event Frames

The simplest method is to create frames out of the events by collecting all events in a specified time window and building them into a traditional frame, an example being shown in Figure 3.10. This method is not without its problems, however, as altering the time window has a large effect on the quality of the frames and it is hard to choose the best value since having a fixed value can mean it is difficult to select an option when the dynamics of a scene can vary significantly, e.g. periods of low motion and high motion as is often the case in gaze estimation. Event frames can also be created by processing events until a threshold number of events have occurred. This, however, will produce very inconsistent frames when motion varies a lot in a scene, as more noise will build up when there is less motion due to a longer time window being covered [102]. This makes it hard to select a good threshold as it needs to be high enough such that the build-up of noise does not reach the threshold before events of interest occur, but low enough to not result in very dense frames when there is lots of motion which will make it hard to decipher areas of interest. This makes it subsequentially unsuitable in many applications, including gaze estimation, due to the varying amounts of motion.

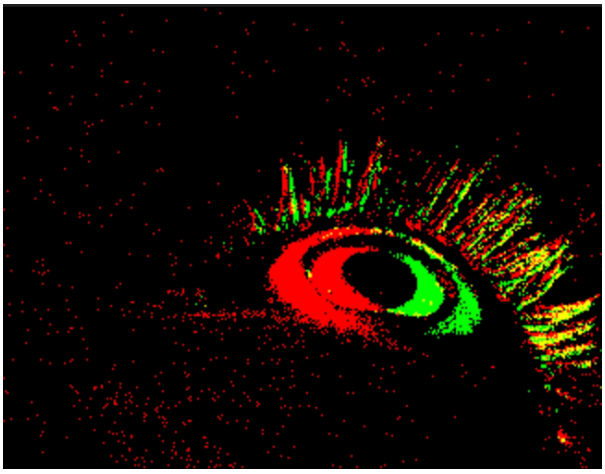

Figure 3.10 Example of an event frame captured over 0.8 seconds where red pixels indicate a positive event, green a negative, and yellow both

There are different methods of collecting events into frames, one of which is to set all pixels where an event occurred to 1 and leave the rest as 0. Typically, event cameras report both

positive and negative polarity events, meaning an increase or decrease in log luminance respectively. To keep all relevant information you choose between using two channels with one channel for each event polarity or use one channel and either differentiate between the two types of polarity, which will be impossible with binary values as there are three possible values (positive polarity, negative polarity, or no event), or use 1 channel and either ignore all events of one polarity, or flatten all events into a single channel that only represents an event occurred and doesn't differentiative on polarity.

Using two channels will possibly create issues with many learning algorithms, as typically when the dimensions of input data is increased, there's an increased chance of issues related to the curse of dimensionality occurring (such as overfitting) [103] [104] [105] [106]. Using one channel with 1 or -1 for each polarity of event is problematic when building the frames for SNNs as is implies more than one spike or timestep must be used to differentiate between the 2, however losing this data by only using binary values on a single channel runs the risk of sacrificing too much information.

Another approach is increasing the resolution of pixels and counting the number of events, setting the magnitude equal to the number of events. This means more active pixels will show up brighter once the values are normalised. This can result in areas of interest being drowned out by random noisy pixels leading to areas of interest being dimmer in the image, as is evident in Figure 3.11.

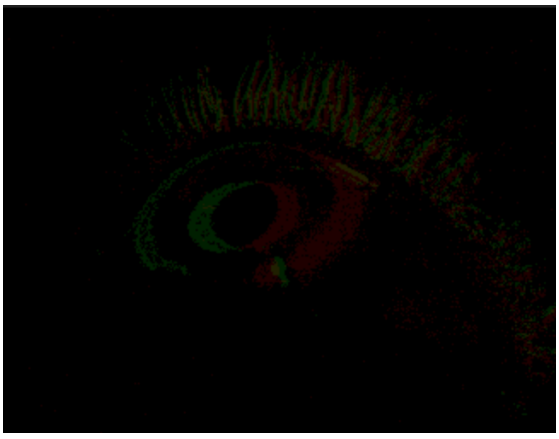

Figure 3.11 Example of an event frame made up by counting events over a 0.8 second window

When using one channel it is important to account for multiple events, of different polarity, occurring at the same pixel. Approaches include using the first or last event in the frame, using the most common event, or counting the events having the final value be the result, e.g., negative event decrements count and positive increments. Again, using an approach such as a count increases in the range of possible values, meaning more complex encoding methods for SNNs are needed to account for this.

3.6.2 Time Stamp Frames

Time stamp frames [107], as seen in Figure 3.12, store the exact timing of events by setting each pixel value to the time in which the event occurred. This maintains more of the temporal precision of the event cameras in the final frame. This, again, will require a more complex encoding scheme. Frames can be normalised to between the desired values, such as between 0 and 1, creating the final frame. This could result in poor frames during static motion, e.g. a stationary eye, when you are relying on older events to build the structure as they will be drowned out by newer events that are entirely noise.

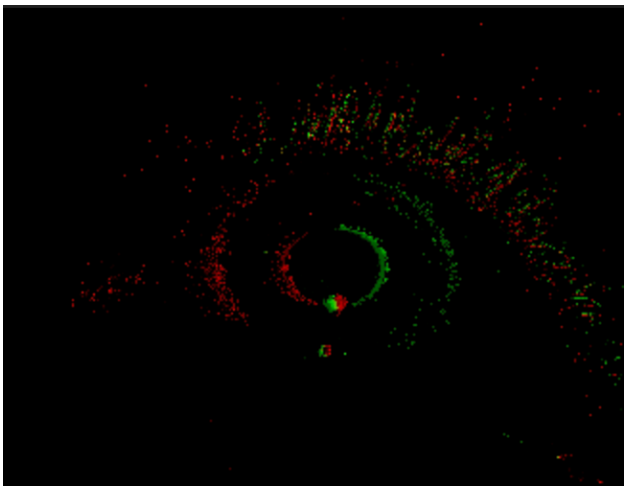

Figure 3.12 Example of a timestamp frame where brighter pixels show a more recent event

3.6.3 Exponentially Decaying Time Surfaces

Time surfaces [108], as shown in Figure 3.13, use decaying values at each pixel where a value is set to the maximum when an event occurs and then drops exponentially over time. This effectively breaks a time window into smaller time windows situated within the original. The value of a pixel describes how recently an event has taken place within that location. Often, space is grouped to create a smaller frame with pixels being grouped with others in their neighbourhood meaning that the final frame is of lower spatial resolution which will be potentially beneficial for efficient computation but runs the risk of being too coarse for gaze estimation which requires precise spatial information to properly predict gaze points. An additional benefit of time surfaces is that it can utilise a continuous event stream by just passing in the frame at the desired intervals for processing.

The formula for an exponentially decaying time surface is:

$$T_i(x,t) = p_i(x) * e^{\frac{T_i(x)-t}{\tau}}$$

[109]

Where $T_i(x,t)$ is the *i*th time surface given for region *x* at current time *t*. $p_i(x)$ is the current polarity of region, *x*, equal to the last event that occurred in the region. $T_i(x)$ is equal to the time the last event occurred in the region and τ is the decay factor. Region *x* can be a grouped neighbourhood of pixels of varying size or can just be a single pixel.

This calculation is done for each region in the frame to create the final time surface, *T,* and can be done over time periods or on the occurrence of an event, i.e., retrieval of a new event. Changing τ influences how prominent older events are in the frame.

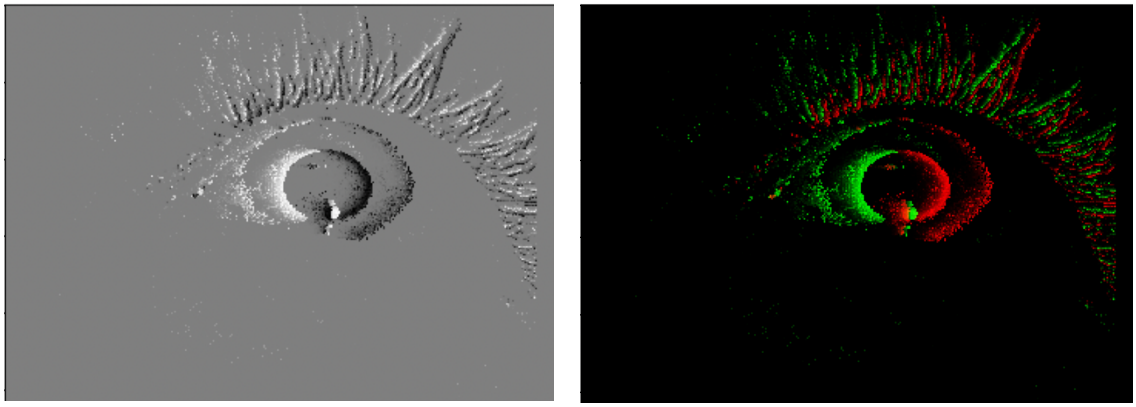

Figure 3.13 Examples of a time surface with neighbourhood size of 1 x 1 with a traditional greyscale (left) and 2 channel RGB (right).

Due to the use of exponential and multiplication calculations, this method is more computationally expensive, and, as such, may not be suited for use in low latency or high frequency processing applications, the aim of this paper.

Time surfaces also struggle with scenes that vary in the amount of motion which, again, is problematic for gaze estimation. The decay factor must be set and the best value for this varies depending on the amount of motion. During high motion periods, this should be low, such that the frame does not become too bloated which would serve to make it hard to pick out meaningful recent events, and during low motion this should be higher such that frames do not have too little data to use to infer meaning. Efforts have been made to combat this with the development of adaptive time surfaces [110] that adjust the decay value on the fly depending on the amount of motion in the scene.

3.6.4 Voxel Grid

Voxel grids, as shown in Figure 3.14, are another common representation often seen in literature [111] [112]. This approach groups parts of frames in both the spatial and temporal domain [113]. If an event occurs, the voxel that the coordinate and time of the event corresponds to is set to 1. This approach creates a 3d frame made up of voxels that stretch in both the spatial and temporal dimensions. The representation maintains more of the temporal resolution by again splitting the time window into smaller sections but loses some spatial

resolution when nearby pixels are grouped. As this is a 3-dimensional representation, however, the same difficulties for processing higher dimensional data apply [103] [105] [106].

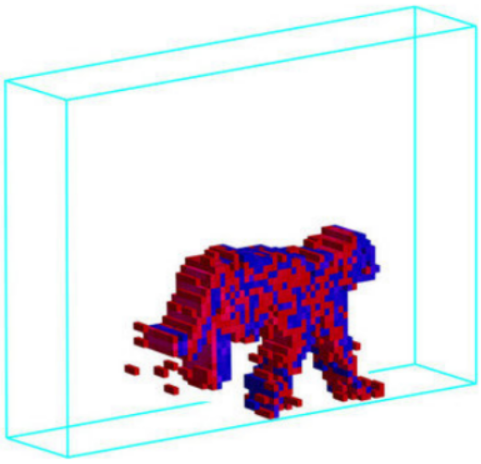

Figure 3.14 Example showing a voxel grid created from event data of a tiger [114]

3.6.5 Transformer Embeddings

The recent popularity of transformer models has also reached into this area of research. A recent approach proposes using a transformer to create an embedding from applying the transformer simultaneously in both the spatial and temporal dimension has been proposed which has shown very impressive results with very low computational cost once trained at as low as 0.51G FLOPs to create an embedding [115]. Although this is a promising area for this application it is avoided for three main reasons. The first, as this is a recent technology other research into this approach is hard to find so the results are hard to back up. The second, this approach requires very large training time and, as hardware resources are limited, this is infeasible. The third, transformer models typically require large amounts of training data, and as event camera-based gaze estimation datasets are limited, this is hard to acquire. As this method creates an embedding it has no meaningful visual structure for a single set of events.

3.7 Event Camera Based Vision

Event cameras have been successfully deployed in many areas of computer vision such as object recognition [98] [116], object tracking [117], and optical flow estimation [118]. Although these applications have shown promising results, they are often tested on specially crafted event datasets [119] [116], also referred to as neuromorphic datasets, which do not necessarily capture the complex intricacies of real-life scenarios.

Event cameras have been used as part of a gaze estimation setup successfully [42] achieving an accuracy of 3.9 degrees at an effective 10,000hz, however this part of a hybrid approach using traditional camera frames as anchor points with events only used to update the gaze prediction in-between the slower traditional frames. This creates an effective 10,000hz frequency but requires the processing of normal frames so cannot be considered power efficient as this is to be avoided when using event cameras for their power efficiency. There have been other approaches at gaze estimation with event cameras using model-based methods, such as using glint detection on event camera data to predict gaze points [120]. There is a deep learning-based hybrid approach [121] that uses event cameras as a method of segmentation on the eye before the segment is processed by a neural network to provide the gaze estimate. This approach achieved cutting edge results with only 30,000 parameters and requiring 1.2 billion floating point operations for a pass through the neural network, which is much lower than other established models, this is three times lower than the 3.6 billion for ResNet-34 which is commonly used backbone architecture [122]. This method, however, requires a lot of additional stages such as the segmentation which require computation and, as such, only runs at 30hz. Additionally, as this is a hybrid approach it still requires the capture and processing of normal camera data.

Currently there is no known purely event camera, deep learning-based approach to gaze estimation.

3.8 SNN and Event Camera

Due to the technologies being very complementary, there have been a lot of attempts to combine the two for computer vision tasks, though this intersection is new and, as such, there are very few established methods. There have been attempts at combining them in areas such as object detection [123], optical flow estimation [124], and object tracking [100]. Additionally, there are attempts at more specialised tasks such as visual angular velocity prediction [125] and video reconstruction [126]. These applications have shown that the technologies can be combined to achieve good results, similar to traditional approaches, in several areas which is promising for its potential use for gaze estimation. Much of this research also highlights the energy efficiency and lower latency of these methods and their suitability for use in areas such as embedded systems and robotics.

Currently there are no known previous attempts at combining the two for gaze estimation.

4 Method

4.1 Analysis of Aims of Research after Literature Review

The aim of this paper was to investigate the use of event camera data and SNNs for low power, low latency, and high frequency processing. This is applied to gaze estimation as an example scenario where this could be beneficial, set out in section 3.1. To investigate this, a deep learning model will be trained on appropriate event representations, detailed in section 3.6, and taking inspiration from methods discussed in section 3.7, to determine if they provide enough information to accurately predict a gaze point. Weaknesses within these approaches will be analysed to aid in the development of an improved method. As using SNNs on neuromorphic hardware to properly see the benefits of this technology was outside of the capabilities of this research, a traditional ANN based model will be used. As discussed in section 3.4, SNN have successfully been created to mimic performance of ANNs, for example with shadow training, so it is hoped proving accurate results can be achieved with an ANN will imply likely success when used with SNNs. To achieve low power, the sparsest representation that can achieve high accuracy will be sought, since it may allow SNNs to process the data more efficiently, also discussed in section 3.4, and provide a better solution than those discussed in section 3.3 for efficient deep learning. Finally, again referring to discussion in section 3.4, representations that can be encoded in fewer spikes, preferably one timestep, into an SNN will be prioritised as this will allow for SNNs with fewer timesteps to be used which will allow for further efficiency but will also allow for low latency and high frequency processing.

4.2 Formalising the Task

Gaze estimation will be undertaken to predict an exact gaze point of a subject providing only visual input, using regression rather than classification to test for higher precision, see section 3.1. This will originally be in the form of standard camera frames, as seen in Figure 5.1, to act as a baseline against approaches using only event data. The aim of this being to match the baseline achieved by the standard camera with the event representations, ergo proving gaze

estimation can be achieved with just event data. Accuracy will be calculated as a mean angle error from test samples so it can be related to literature as discussed in sections 3.1 and 3.2.

4.3 Dataset

Although gaze estimation datasets from traditional cameras are plentiful, only one dataset could be sourced for gaze estimation with event cameras, the 'Event Based, Near Eye Gaze Tracking Beyond 10 000hz' dataset. As such, this dataset is used exclusively within this paper. This near eye gaze dataset consists of videos from both traditional and event cameras captured at the same time of users' eyes as they look at points highlighted on a 40-inch, 1980 x 1080 screen, 40 centimetres in front of them. Users' eyes are positioned to be approximately in the centre of the screen, with the field of view of all points shown to users covering 64 degrees vertically and 96 degrees horizontally. There are two modes of stimuli used during data capture, the first of which is saccade movements where consecutive highlighted points are disjointed and often with large distance between them. In this mode a new point is highlighted every 1.5 seconds, with the user then moving their focus to the new point. The second mode is tracking, where points move in a continuous motion and users track the highlighted point around the screen.

Both cameras run at a 346 x 260 pixel resolution, with the traditional camera capturing at 25 frames per second in greyscale and the event camera, the Davis 346b camera by iniVation, having a temporal resolution of approximately 20 microseconds.

Data was collected from 27 different subjects with videos captured separately for both eyes. Image data is labelled as [index, row, column, stimulus, timestamp], where the index indicates the order images were captured, row and column are the y and x coordinate of the current stimulus, stimulus is the current mode (saccade, tracking or stop for none) and the timestamp is the time in microseconds since data capture started. Event data is stored in the AER data format as [polarity, timestamp, row, column] where the polarity is the polarity of the event (0 for negative, 1 for positive), timestamp, is again, the timestamp in microseconds since the start of data capture and the row and column are the y and x coordinates of the current stimulus. Periods in the event data can be matched to the traditional frames using the timestamps since they are calibrated to the same start point.

4.3.1 Processing the Dataset

The dataset in its raw format was not suitable for use in the task, but as it was the only dataset that could be used for this experiment, it was determined that it must be processed to make it suitable. As such, the following steps were taken to process the data so it can be used for the current task. Only one eye was used for gaze estimation to lower both the training time and storage requirements compared to using both eyes. Initially frames were selected from the traditional camera data, from which timestamps can be collected to act as anchor points in the event data. Events can be collected from before these points to create event representations with events preceding a known gaze point.

First, as images are labelled using the current location of stimulus and not the position in which the user is looking, they do not always align. In the case of the tracking mode, this is not a problem as the stimulus moves to neighbouring pixels slowly so the label will always be very close to where the user is looking. The saccade mode, on the other hand, has large distances between stimuli, and although eyes can move fast, the user's response time to the stimuli must also be accounted for. Resulting in a period between when the new stimulus is shown, and when the user is looking at the point, as shown in Figure 4.1. Although the saccade portions of the dataset could be removed to account for this, it was deemed important to maintain this data for two primary reasons. One, eyes when stationary, have unique properties and it is important to be able to track them both during motion and when stationary, this is more apparent when looking at event data than traditional frame data due to the issues with stationary scenes as mentioned in section 3.5. Two, the dataset is already limited in size so reducing it further is ill advised. To remedy this, first the data was observed, and by the 0.5 seconds mark of the 1.5 second period between saccade stimuli, users had reached the new position and were looking at the labelled stimulus point. To avoid what would be essentially duplicated images in the dataset, only the last frame before a new stimulus is shown, and the frame at the halfway point of the 1.5 seconds between stimuli was extracted. The 0.5 seconds to complete motions served as an important point for the event data as this leaves 1 second where the data is exclusively of a stationary eye. Taking these two frames means event data will capture both fast motions before the frame at the halfway point, and low/no motion before the frame at the end of the stimuli period, as demonstrated in Figure 4.1.

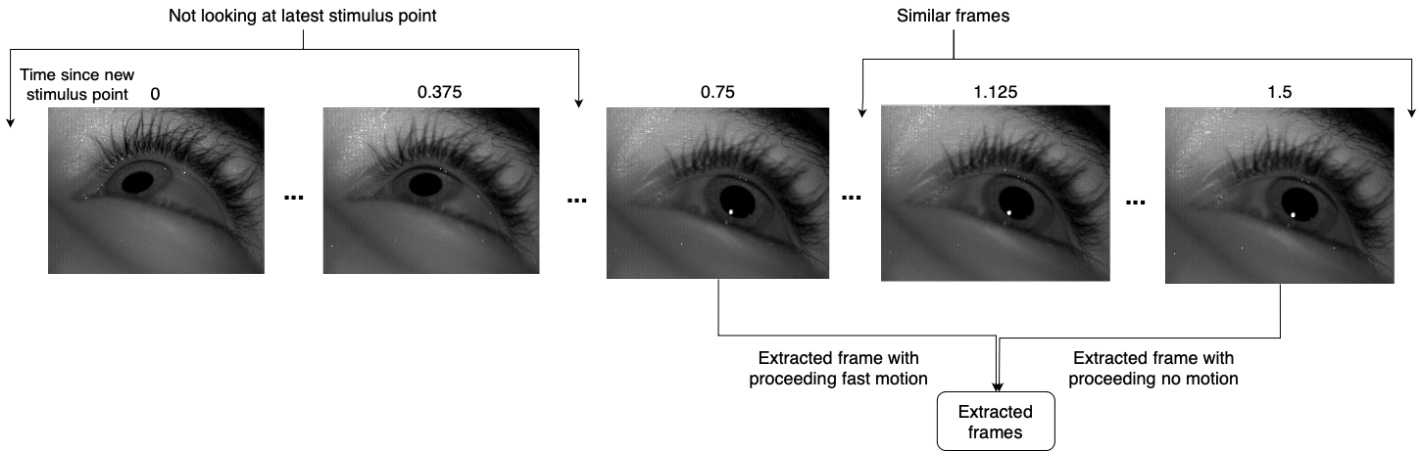

Figure 4.1 Diagram showing which frames were extracted from video

This meant that for each user, there were 240 frames. In addition, 120 random selections from the tracking portion of the dataset were added, for a combined total of 360 frames. When looking at event data around the timepoints of these frames, there would be an even distribution of fast motion, no motion, and moderate motion for the tracking frames.

Two subjects, User 1 and User 2 had incomplete data. As users will be randomly selected for train, validation and test sets, these users were removed as this could create inconsistency if these users are selected for different sets across experiments by changing the ratio of test to train set size. Additionally, it was observed that User 4 had glasses on during the experiment. Although this is an important characteristic any production gaze estimation system should be able to accommodate, as this user was the only participant with glasses on and the train/test sets will be made by splitting the users randomly, User 4, with the glasses, cannot be present in both sets which could create inconsistent behaviour, so this user was removed leaving 24 users in the final set.

This created a final total dataset size of 8640 time points. Two users were selected at random (User 12 and User 18) to act as the test set that is unseen during training for each model and used to evaluate the model's accuracy, creating a train/test split of 7920 to 720.

4.4 Performance Metrics

It was decided, that to determine success of meeting the aims set out in section 4.1, the following metrics would be used considering both the properties of the representations and the performance of the models that were trained with them:

- **The amount of non-zero pixels** – Representative of the number of spikes that would be required to encode a frame into an SNN as specified in section 4.4.2. Higher sparsity with less non-zero pixels will mean potentially better power efficiency with SNNs.
- **Whether values are binary** – Binary images can be encoded into SNNs in a single timestep, opening the possibility for more efficient, lower latency, and higher frequency models. Therefore, this a favourable property defined as true/false.
- **Accuracy of the model** (in mean degrees error) – As specified in section 4.4.1. The model should be as accurate as possible. The frames with the worst prediction accuracy from the test sets will also be shown to help identify what types of input the methods struggle to deal with. There's no point practical benefit in more efficiency, lower latencies and higher frequencies if accuracy cannot be maintained.

4.4.1 Accuracy

For calculating accuracy as an angle, first the x and y coordinates are converted to a gaze angle for both the prediction and the true values. To do this, the distances in millimetres from the centre of the users' gaze are calculated using:

$$d_x = (p_x - m_x) * (\frac{W}{Xp})$$

$$d_y = (p_y - m_y) * (\frac{H}{Yp})$$

[127]

Where d_x and d_y are the distance in millimetres in the x and y direction respectively from the centre point. p_x and p_y are the x and y pixel coordinates of the distances being calculated. m_x and m_y are the midpoint in pixels in the x and y directions. *W* and *H* are the

width and height of the screen in millimetres. Xp and Yp are the total number of pixels horizontally and vertically.

For the case of the study set-up used for this dataset, the screen resolution is 1920 x 1080, *Xp* = 1920 pixels and *Yp* = 1080 pixels. The size of the screen is 40 inches, *W* = 878 millimetres, and *H* = 494 millimetres. The user is in the centre of the screen, m_x = 960 pixels and m_y= 540 pixels.

After the distances have been calculated, the gaze angle in both the horizontal and vertical direction can be calculated as:

$$\theta_x = \tan^{-1}(\frac{d_x}{D})$$
$$\theta_y = \tan^{-1}(\frac{d_y}{D})$$

[127]

Where *D* is the distance of the user from the screen in millimetres, which is 400 millimetres in this set-up. θ_x and θ_y are the gaze angles in the x and y direction and are calculated in degrees. This is calculated for the true x and y coordinate, and the predicted coordinate. Providing θ_x and θ_y for the true, and $\hat{\theta}_x$ and $\hat{\theta}_y$ for the predicted angles.

The overall angle error can be calculated using the l2 norm of the difference between the predicted and actual gaze angles:

$$A = \left\|(\theta - \hat{\theta})\right\|$$

[44]

Where A is the final accuracy in degrees and $\theta = (\theta_x, \theta_y)$ and $\hat{\theta} = (\hat{\theta}_x, \hat{\theta}_y)$.

4.4.2 Number of Spikes

The number of spikes required to encode all values within an event representation into an SNN is dependent on both the number of non-zero values and the encoding scheme used; therefore, the encoding scheme must be defined. For binary values, binary encoding will be considered, and for non-binary values, time-to-first-spike. Both require a single spike at maximum for any value, but non-binary values encoded using time-to-first-spike can't be encoded in a single time step and, as such, impose a greater than 1 timestep.

For event data, the number of spikes required for input is equal to the total non-zero values in the input. This can be calculated using:

$$\sum_{x}\sum_{y} 1 : p_{xy} > 0$$

Changes in sparsity are reported change in the number of non-zero values that require spikes.

4.5 Model

As the choice was made to train an ANN for testing event camera data representations, a suitable model backbone needed to be selected. As discussed in section 3.2, most existing approaches to gaze estimation use CNNs, so this was the chosen architecture type. As the intention, as specified in section 4.1, is to prove success on an ANN with the hope an SNN can mimic its performance, a model would be selected where SNNs had already successfully achieved this on another task [128]. As the focus is on creating good input representations that can potentially achieve the power efficiency targeted in section 4.1 in the future, and not creating a specific efficient model, the choice was made to not focus on choosing a smaller more efficient model during this research and instead choose a model with a proven track record of success that would provide the best results, removing as much ambiguity of the model

selection on results as possible. This resulted in the ResNet architecture being selected for the backbone of the model.

4.5.1 RESNET

Residual networks have been deployed for image processing tasks for many years, since their introduction in 2015, to great success [122]. This type of network was originally proposed to aid in training very deep neural networks by providing a solution to help against the vanishing gradient problem [129] [122]. The ResNet family of models are CNNs that make use of multiple convolutional layers, grouped into pairs forming residual blocks, before fully connected layers. Residual networks come in many forms with different numbers of layers that affect the total number of trainable parameters in the model. The most common sizes being, 18, 34, 50, 101 and 152 layers where two layers are used for input and output, and the rest are convolutional layers. ResNet models in different sizes have successfully been applied to gaze estimation, including achieving 6 degrees accuracy on a difficult feature based gaze estimation dataset with ResNet-50 [130], beating the accuracy of models such as AlexNet with ResNet-18 [131], and has been combined with other techniques such as attention mechanisms to achieve an accuracy of around 4 degrees on some datasets [132]. Due to this, ResNet-34, as shown in Figure 4.2, was selected as a balance between size and performance with the hope that this would remove as much ambiguity around the effect of the model choice on the results.

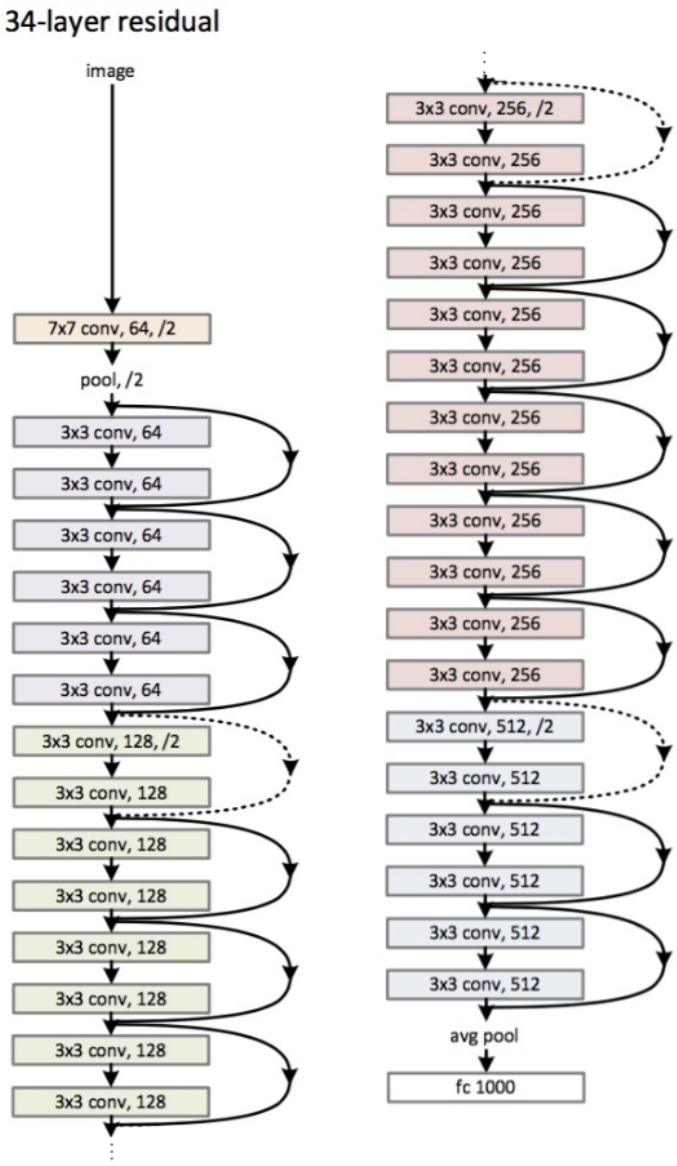

Figure 4.2 ResNet-34 model architecture [133]

4.5.2 Original Model Architecture

The model had to be adjusted for use with resolution of the input images from the dataset. Furthermore, during initial testing, it was discovered that adding additional fully connected layers increased performance of gaze predictions, so they were included, to give a final architecture as seen in the simplified diagram in Figure 4.3, with a total 30,194,882 parameters.

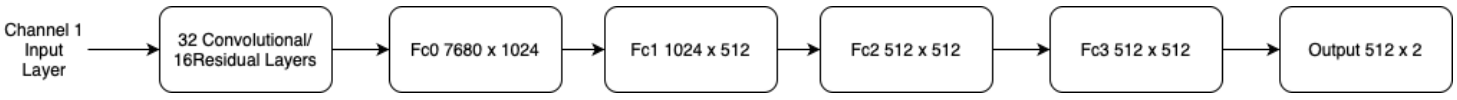

Figure 4.3 Simplified model of the adjusted ResNet-34 architecture utilised during testing

4.6 Model Training

Various representation creation methods were used to assemble datasets for training models to discover the performance of existing methods. This will be utilised to see the current benchmark they set and help diagnose problems with current event processing methods to help guide the development of a new method. Each model was trained using an ADAM optimiser with learning rates of 0.0005, 0.001 and 0.005, with the best accuracy reported. The loss function used was the mean squared error (MSE) using the Euclidean distance between the actual and predicted xy gaze point coordinate vectors. MSE was selected as it punishes low precision as the effect on loss increases exponentially as a prediction moves further away from the actual values. This will encourage the favouring of the majority being closer to their target rather than having some very close predictions with very far away predictions valued the same, as would be the case when using the mean absolute error. This is since, in most applications of gaze estimation, precision will be a favourable characteristic. Data augmentation was used on the images to reduce overfitting and improve test accuracy, but as many common augmentation techniques would not be suitable for this task due to changing the image in a way that would make it impossible to predict the correct gaze point consistently, e.g. flipping the image, this was limited to just translation and scale. This was applied randomly with a probability of 0.8 to each image at every epoch. A batch size of 64 was used and training was for 100 epochs each time. This took approximately 8 hours to run on a M1 MacBook Air and around 1-2 hours on the Google Colab V100 GPUs.

All training was completed with a train and validation set. Two users were chosen at random from the training set for use as validation during training. The final model was the model in state at the epoch that produced the lowest validation error.

Training performance was monitored by plotting training and validation losses, see Figure 4.10 for an example, for each epoch to detect any strange behaviours. Additionally, visualisations of predictions on the test datasets were created to verify results as seen in Figure 4.4 and the frames of the two worst and two best prediction accuracies from the test set were displayed for each model, see figures 5.3 and 5.14 for examples. The aim of which being to help diagnose

what frames perform well and what do n0t across the models to guide the development of the new approach seen in section 6.

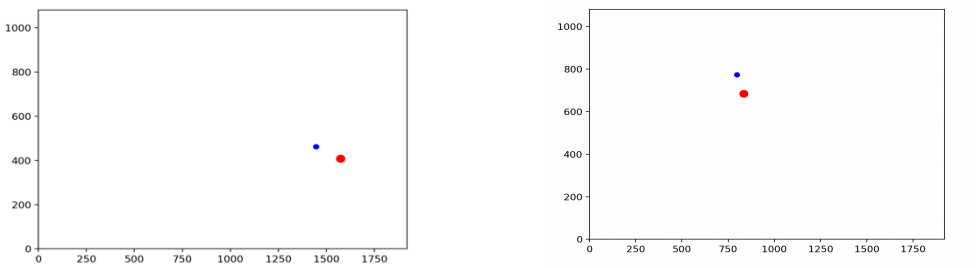

Figure 4.4 Example visualisation of gaze prediction where the red dot represents the actual gaze label and blue the prediction. 1920x1080 graph represents screen users were looking at

5 Results of Existing Methods

5.1 Choosing Methods to Test

As seen in section 3.6, there are several methods of creating event representations. Before testing the suitability of these methods had to be assessed for gaze estimation. This was completed to direct testing towards methods that met the requirements of the task, or to provide meaningful insights.

From the methods discussed within section 3.6, time surfaces can be excluded due to the computational cost of their generation which is attributed to expensive exponentiations and multiplications for pixel value calculation. Instead, the effect of temporal resolution in pixel values on performance will be tested using timestamp frames.

Voxel grids were avoided as they must have large 3rd dimensions to maintain the temporal resolution. As 3-dimensional images are problematic for training models with a limited dataset size due to overfitting [105] [106], as mentioned in section 3.6.4, the channels would likely require separate processing. Although the slices making up the 3rd dimension will hold information in an appropriate form to be processed separately, it will effectively be smaller time windows with less spatial information, which is made redundant since the effect of smaller time windows will already be assessed by changing the time window of normal event frames.

This left the various methods of creating event frames and timestamp frames as the remaining candidates. These methods have variations on their implementation that affect performance which raised three questions, for which testing was performed to attempt to answer:

- What is the effect of using different methods of handling events of different polarities?
- What is the effect of using different methods of assigning values to pixels?
- What is the effect of changing the size of the time window?

The best performing methods are carried forward at each stage, using a greedy method to search for the best current method.

5.2 Traditional Camera Frames

A model was trained on the standard camera frames, creating both an accuracy baseline for event data to be compared against and verifying the model with the selected parameters can achieve close enough to state-of-the-art accuracy on the dataset to show there are no major issues. Images from the dataset were used without modification as shown in Figure 5.1.

Figure 5.1 Example of traditional camera frame

Representation method	Number of non-zero pixels	Binary valued	Accuracy (degrees)
Traditional Frame	88,723	False	5.99

Figure 5.2 Table showing results of test on traditional frames

The results in Figure 5.2 show that this model was able to achieve a 5.99 degrees test accuracy. Although this is lower than some accuracies seen in the literature, it is only around 2 degrees off of the state of the art, meaning approximately a 50% increase in average error. This discrepancy is to be expected given the small size of the dataset which will limit the model's ability to learn and generalise to the test set, alongside only using a single eye rather than both which is common in the literature [38] [39] [134]. These restrictions will be the same for both the standard camera frames and event representations, so the baseline was deemed acceptable. For these images, almost every pixel is non-zero with an average of 88,723 non-zero pixels per image. Therefore, the required number of spikes for these images is very close to the maximum possible which is 346 x 260 = 89,960. The values of pixels are non-binary so more than one

timestep is required to encode them. As they are 8-bit depth images, each pixel has 256 possible values meaning that at least 256 timesteps will be required to keep full resolution.

As can be seen in the worst accuracy images in Figure 5.3, the traditional camera frame model cannot handle blinks well as accuracy is very poor when a blink is captured, with worst case accuracies of 45.4 and 36.9 degrees error both being from captured blinks.

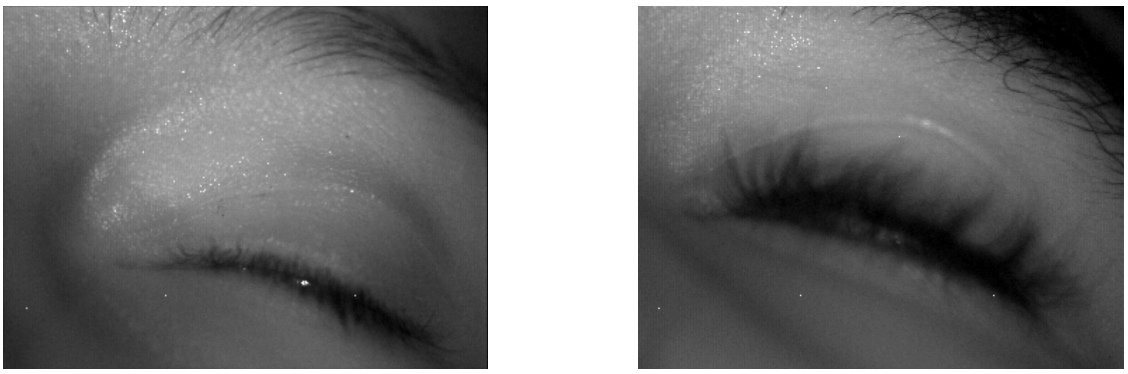

Figure 5.3 Images that had the highest error from the test set, with 45.4 degrees (left) and 36.9 degrees (right) error respectively.

5.3 Event Based Methods

After the baseline, the next step was training models using event data. Tests were designed in a way to answer the questions about using event data as set out in section 5.1.

5.3.1 Handling Event Polarities

The first issue that was tested was how to represent events of different polarities. Originally, three methods for this were deemed as possible solutions from those discussed in section 3.6.1. First, ignoring events of one polarity and creating a one channel frame with the events from the other remaining polarity. In this implementation only positive events were kept. Second, flattening all events into one channel that holds events of both polarities. Finally, using two channels with one for each polarity, positive in the first and negative in the second. Although using two channels would create a 3-dimensional image that has been discussed as an issue in sections 3.6.1, 3.6.4, and 5.1, this was carried out as verification and to test if the benefits of separating the polarities could still provide increased performance. Examples of these frames

can be seen in Figure 5.4. These frames were created with a 0.8 second time window as a starting point, meaning events from the types of motion as defined in section 4.3.1 would not overlap into multiple frames. This exact value was chosen based on manual observations of results from trying values based on how well a random selection of frames showed the features of the eye. Pixels were assigned binary values where if any relevant events occurred at the location during the time window, it was set to 1.

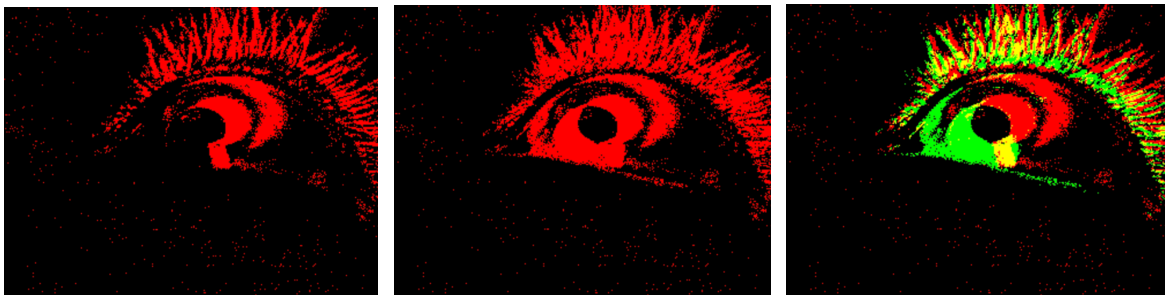

Figure 5.4 Example event frames made by only counting events of 1 polarity(left) flattening all events into 1 channel (centre), and using 2 channels 1 for each polarity (right)

To accommodate training the current model using a two-channel input for the separate polarity channels method, the input channels of the first convolutional layer had to be increased. This meant this model would have a slightly increased 30,198,018 parameters.

Method used to create frame	Number of non-zero pixels	Binary valued	Accuracy (degrees)
Traditional Frame	88,723	False	**5.99**
Only positive polarities	12,965	**True**	9.21
All polarities in 1 channel	**10,369**	**True**	7.98
1 channel for each polarity	18,630	**True**	8.21

Figure 5.5 Table showing results from different methods of handling event polarities.

The results in Figure 5.5 show, none of the methods matched the performance of the baseline in terms of accuracy, however they were within approximately 2-3 degrees whilst ranging from 79-85% improved sparsity. Although the accuracy isn't at baseline, achieving so close with the increased sparsity is a promising start.

As expected after the discussion in section 3.6.1, the model, trained on 2 channel input, showed signs of overfitting with diverging training and validation losses over time as seen in Figure 5.6. However, as seen in Figure 5.5, the test accuracy was still in line with that of the other methods which did not show overfitting. The final average absolute pixel difference for the training predictions with the two-channel model got as low as 82.5, whereas this was 143.7 when putting all polarities in one channel, which achieved a better test accuracy despite the higher train loss. This suggested that there could be benefit in maintaining differentiability of event polarities, but the overfitting needed to be addressed, which if completed was theorised to potentially lead to better test accuracy.

It was noted, that although with less accuracy, the model was able to learn based on just events of one polarity. This meant that a model could be created that could process the two polarities separately as single channel frames and find meaningful information that could be used to provide a single output. Considering this, it was deduced that it could be a method of handling the overfitting of two channel input whilst allowing a model to make full use of the information gained from keeping both polarities separate.

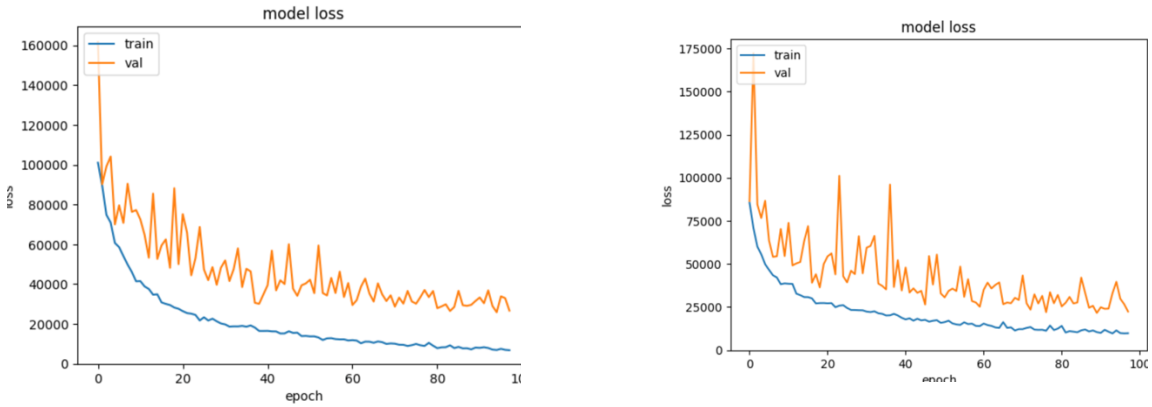

Figure 5.6 Graphs showing training and validation losses after each epoch during training for 2-channel model (left) and 1-channel model (right), Note graph on left validation loss is starting to diverge, on the right difference is more consistent.

5.3.2 Modified Model for Processing Both Polarities

Two model structures were possible for processing the two channels separately. Either creating two models, one for each polarity and taking the average of the two predictions as the final output. Alternatively, creating a model that would at first process the two channels separately

before being combined later in the model to allow the model to find meaningful connections between features in both polarities. The second structure was used as the occurrence of events in different polarities intrinsically have relation to each other, and using one model will allow for this relation to be used to create better predictions.

A modification of the existing model was designed where the polarities would be passed into the convolutional layers separately. The output of the fully connected layer after the convolutional layers were combined before being passed onto further fully connected layers. This would allow features to be detected separately in the convolutional layers, to hopefully decrease overfitting.

To counteract the increase in parameters from having two sets of convolutional layers, the number of kernels in the final three residual blocks were reduced to 256 from 512, and the fully connected layers after the convolutional layers had their output dimensions reduced to 512, meaning once both were combined into a single vector, the vector would have dimension of 1 x 1024. One fully connected layer was also removed to further reduce trainable parameters.

These modifications resulted in a slightly smaller model with 28,534,274 parameters, compared to the original 30,194,882. The simplified architecture of the new model can be seen in Figure 5.7.

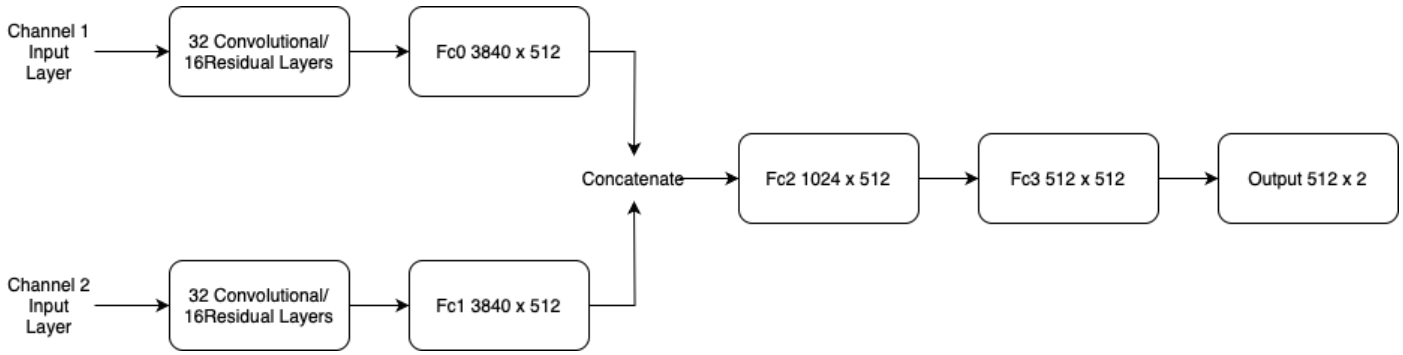

Figure 5.7 Modified architecture used to process 2 channel input. Based on ResNet-34

This modified model was then trained using the frames with two separate polarity channels. As seen in Figure 5.9 there was a large 1.7 degrees accuracy increase with this model. Additionally, when looking at the graph of training and validation loss in Figure 5.8, there was no longer a large gap between the two. Apart from the outlier epoch training loss in the middle of training, the graph showed steady improvement to both training and validation loss that was not diverging. This shows that processing types of polarity separately is a valid approach leading to increased accuracy, whilst using a similar number of parameters. This was the preferred method moving forward due to its accuracy being within 0.52 of baseline.

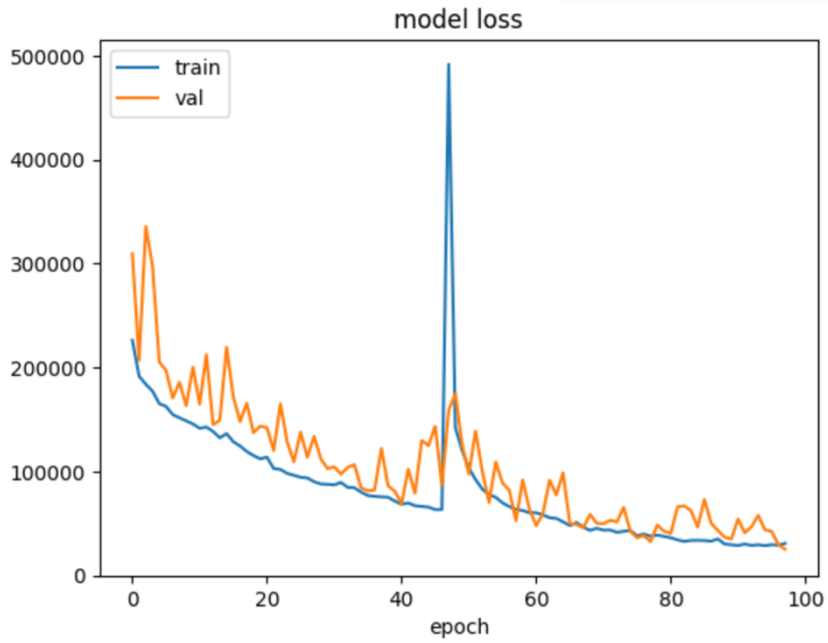

Figure 5.8 Graph showing training and validation loss of modified model. Note strange spike in training loss during training, which was deemed as outlier behaviour, and as such unimportant, due to non-repeatability

Method used to create frame	Number of non-zero pixels	Binary valued	Accuracy (degrees)
Traditional Frame	88,723	False	**5.99**
Only positive polarities	12,965	**True**	9.21
All polarities in 1 channel	**10,369**	**True**	7.98
1 channel for each polarity	18,630	**True**	8.21
2 channel modified model	18,630	**True**	**6.51**

Figure 5.9 Table showing results for different methods of representing event polarity with the new model included

5.3.3 Assigning Values to Pixels

Next was how to assign values to pixels in the frames on occurrence of events. For this the binary two-channel image from the previous section was compared against time stamp frames and frames built using the count of events at each pixel coordinate. Again, these frames were created with a 0.8 second time window. All frames were normalised to between 0-1 for visualisation. All frames used two channels for polarities and utilised the new modified model described in section 5.3.2.

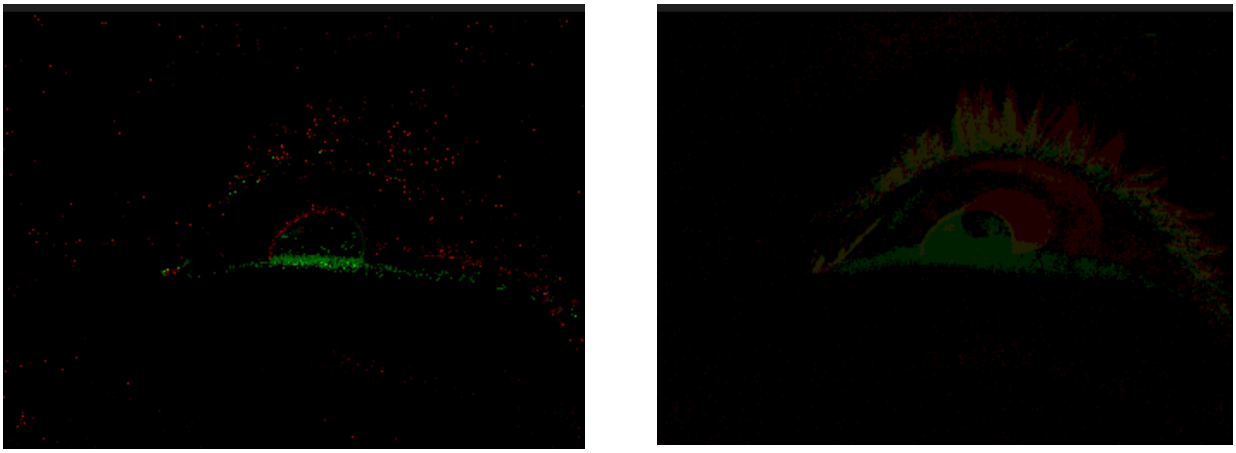

Figure 5.10 Example timestamp frame (left) and event count frame (right)

Method of assigning values to pixels	Number of non-zero pixels	Binary valued	Accuracy (degrees)
Binary	18,630	**True**	**6.51**
Event count	18,630	False	8.12
Timestamp frame	18,630	False	9.67

Figure 5.11 Table of results from different methods of assigning values to pixels based on events.

Interestingly the results in Figure 5.11 show that increasing the temporal resolution with timestamp frames degrades performance significantly. This could be as CNNs are not designed to handle temporal information. As can be seen in Figure 5.10, the pixel intensities on average, become duller. As CNNs use the input value multiplied by a weight to detect visual structures, it is possible that older events that have the lower intensity are failing to fire neurons meaning features including them are not getting detected. The areas of interest for identifying the eye

being duller at a given time point shows that older events are needed for detecting the structure of the eye and relying on newer events that could just be noise does not provide information on the position of the eye. This will be particularly prevalent in periods of low or no motion as it would be necessary to go back further to build enough information to detect features. Additionally, counting events also seems to have a negative effect on performance. Again, noise could be responsible for this by drowning out areas of interest with the occurrence rapid repeat events, which was discussed in section 3.5, which will cause noisy pixels to show up with more intensity than areas of interest. Based on these results, the binary option was chosen as the best as it has both the best accuracy and can also be encoded in a single timestep.

5.3.4 Time Window Size

Finally, the effect of changing the time window was investigated. For this the binary method that produced the best results in the previous sections was used again, but events were collected over a time window of 0.2, 0.5, 0.8, and 1.1 seconds, shown in figure 5.12.

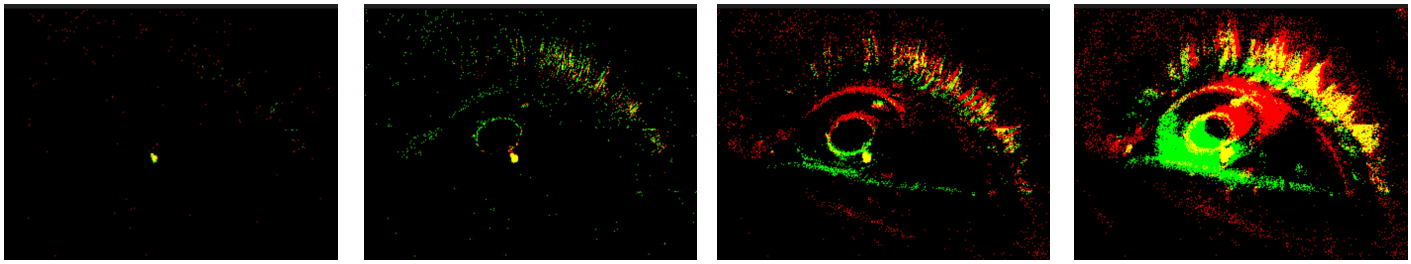

Figure 5.12 Example images of 0.2 (left), 0.5 (centre left), 0.8 (centre right) and 1.1 (right) second time windows from the same timepoint

Time window length (seconds)	Number of non-zero pixels	Binary valued	Accuracy (degrees)
0.2	**3,746**	True	10.90
0.5	10,535	True	7.64
0.8	18,630	True	**6.51**
1.1	24,389	True	6.83

Figure 5.13 Table showing results of different length time windows

As can be seen from the results in Figure 5.13, accuracy increases with extended time windows up to 0.8 seconds, even though increasing the time window increases the chance of capturing

a blink. As unlike in a traditional camera, where the blink would have to happen at the exact moment of capture to have an effect, a blink happening at any time of the time window with an event camera will affect the produced frame. This was theorised to be a problem for event frames, but benefits of the increased window seem to outweigh this. When looking at the frames that caused the lowest accuracy in Figures 5.14, 5.15, 5.16 and 5.17, for the shorter 0.2 window, frames that appear to be just noise with no structure of the eye are the worst performing. For the longer windows all the worst frames contain what appear to be blinks, or periods of extreme motion. This shows that too few or too many events are both bad for accuracy and that blinks do cause bad accuracy. Although longer windows are more accurate, they are, as expected, less sparse, so the accuracy comes at a cost. For example, increasing from 0.5 to 0.8 decreases sparsity by 43% for a 1.13 improvement in accuracy, making the decision of the best time window less obvious. Part of the reduced sparsity comes from additional noise collected at longer windows and not just events caused by the eye, as is evident in Figures 5.19, 5.20, 5.21 and 5.22.

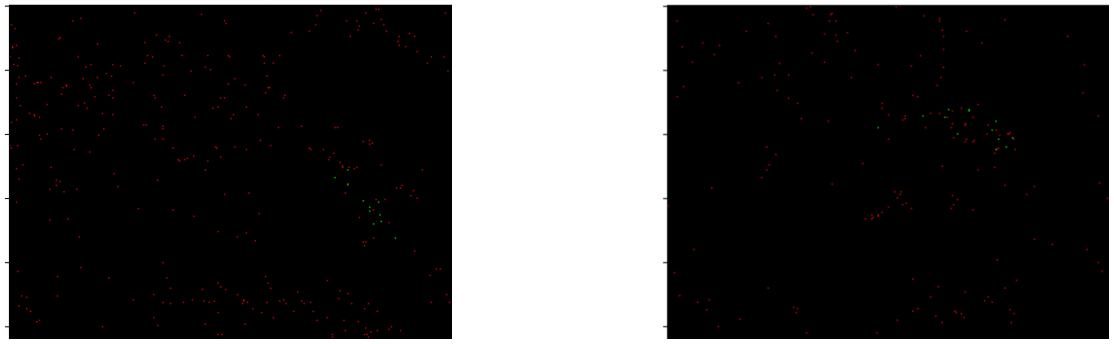

Figure 5.14 Worst performing images for 0.2 seconds with 66.4 (left) and 64.3 (right) degrees error.

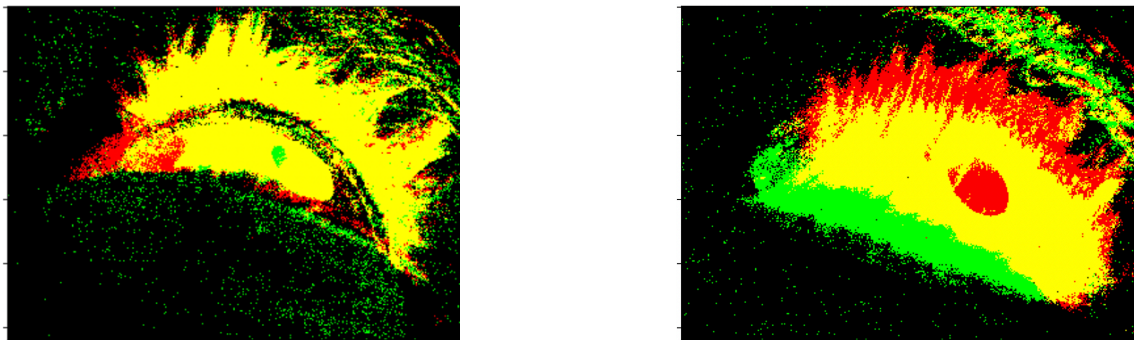

Figure 5.15 Worst performing images for 0.5 seconds with 84.6 (left) and 61.4 (right) degrees error.

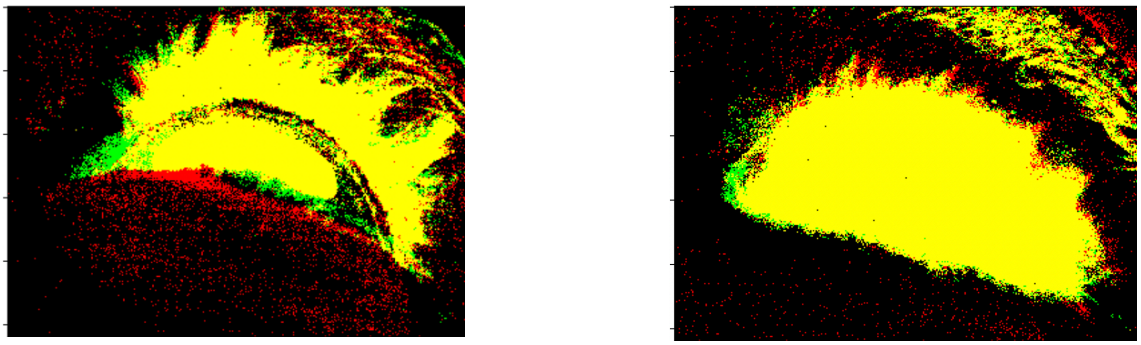

Figure 5.16 Worst performing images for 0.8 with 42.4 (left) and (right) 38.9 degrees error.

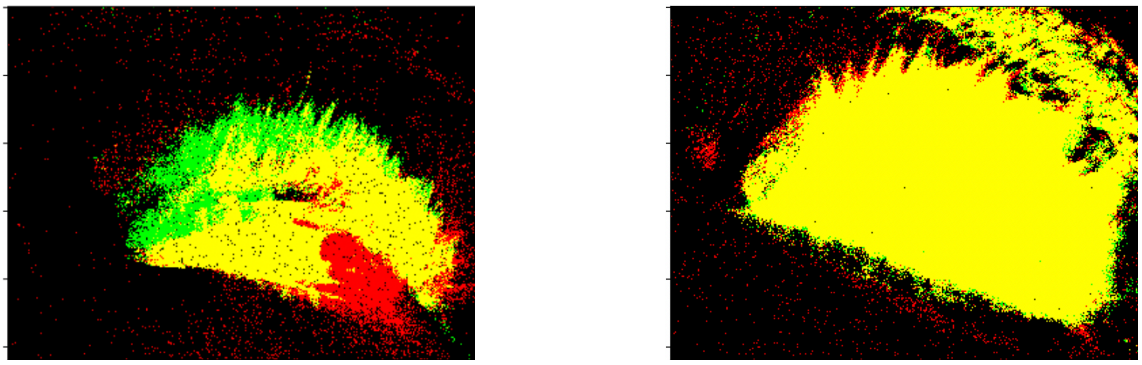

Figure 5.17 Worst performing images for 1.1 seconds with 52.3 (left) and 47.6 (right) degrees error.

Accuracy was then split by the types of motion, discussed in section 4.3.1, and reported separately. As can be seen in Figure 5.18, for periods of no motion, accuracy increased with time. However, for slow tracking and fast motion, there was a dip in the 0.5 and 0.8 window with worse performance on either side. This shows that longer periods are beneficial to increase accuracy during stationary motion, possibly due to being able to build up enough events from microsaccades of the eye to locate it. During fast motion, lower time windows are beneficial to a point but can be too low. Another possibility for poor performance at 0.2 seconds is the quality of frames are so poor during stationary motion, to the extent that it effects the results of other types by completely disrupting learning of the model. Of possible note is the slow motion having the best accuracy for all windows apart from a slightly worse performance with 0.5 seconds suggesting frames are generally better quality when built up with steadier motion. Investigating further by looking at images that produced the best performance in each category, as seen in Figures 5.19, 5.20, 5.21 and 5.22, the best results are when all key features of the eye can be clearly located without lots of events obscuring them for all types of motion.

Time window (secs)	Stationary motion accuracy (degrees)	Slow tracking accuracy (degrees)	Fast motion accuracy (degrees)
0.2	13.79	8.30	8.71
0.5	9.20	6.90	**6.81**
0.8	6.61	**6.02**	6.89
1.1	**6.20**	6.62	7.68

Figure 5.18 Table showing results of different length time windows split by type of motion in the scene

Although accuracy is higher at 0.8 seconds, there are disadvantages to using a higher time. Primarily, latency is limited by the time window if duplicate processing is to be avoided. For example, using an 0.8 second time window, if events aren't processed multiple times, one frame is produced every 0.8 seconds. The obvious solution is to have overlapping time windows, however the whole time window would have to be processed separately for each frame meaning that all events in the overlapping sections will be processed multiple times. When the window is longer this means more duplicate processing. Due to these drawbacks, it was decided that a new method would be needed to find a balance between efficiency and accuracy.

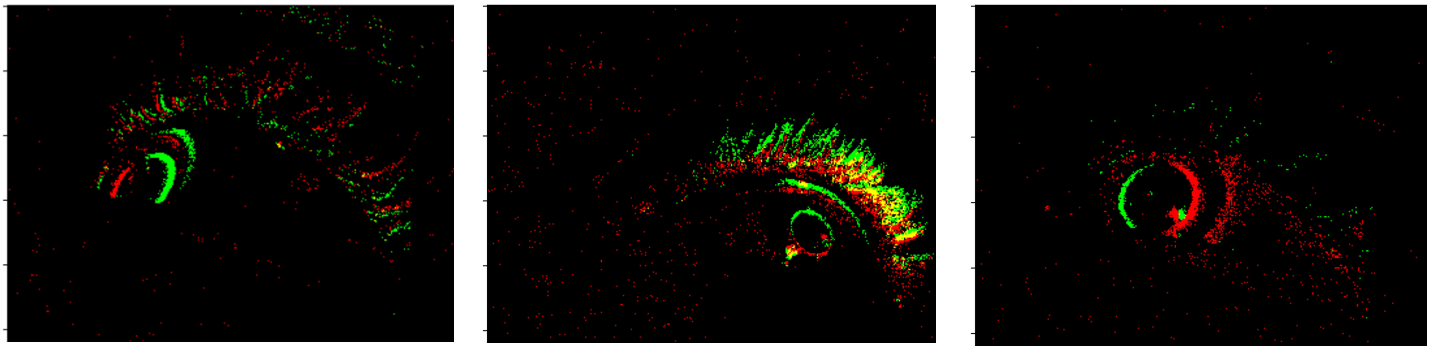

Figure 5.19 Best performing images for 0.2 seconds for stationary with 0.76 error (left), slow tracking with 0.35 error (centre) and fast motion with 0.45-degree error (right)

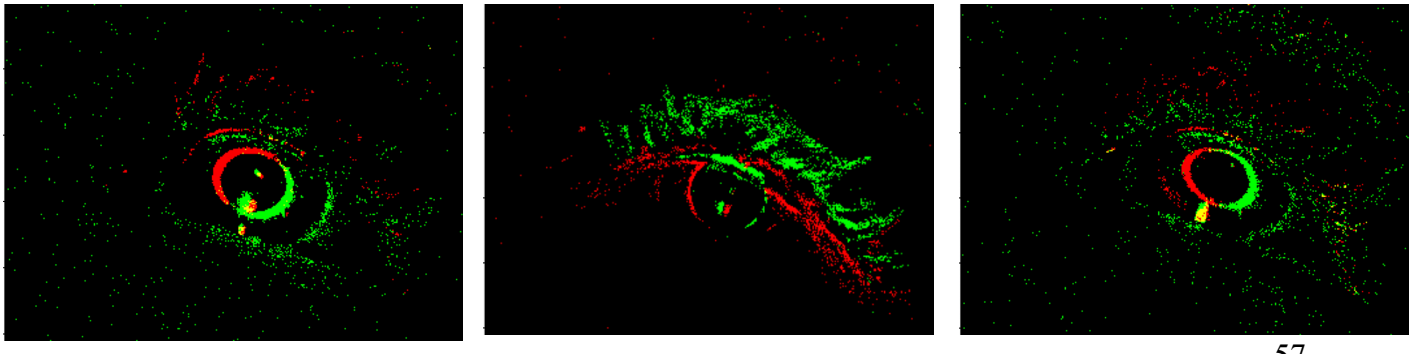

Figure 5.20 Best performing images for 0.5 second time window for stationary with 0.54 (left), slow tracking 0.52 (centre) and fast motion with 0.65 (right) degree errors

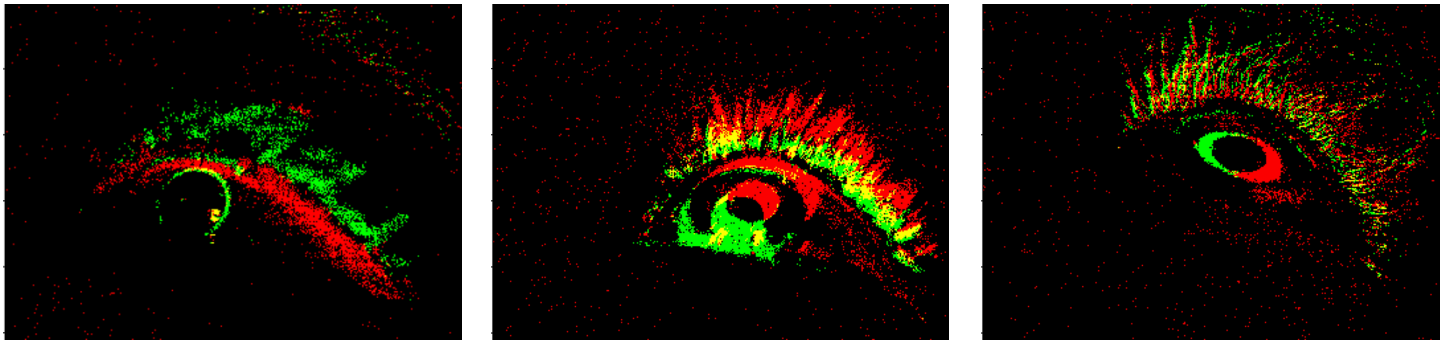

Figure 5.21 Best performing images for 0.8 second time window for stationary with 0.25 (left), slow tracking 0.32 (centre) and fast motion with 0.37 (right) degree errors

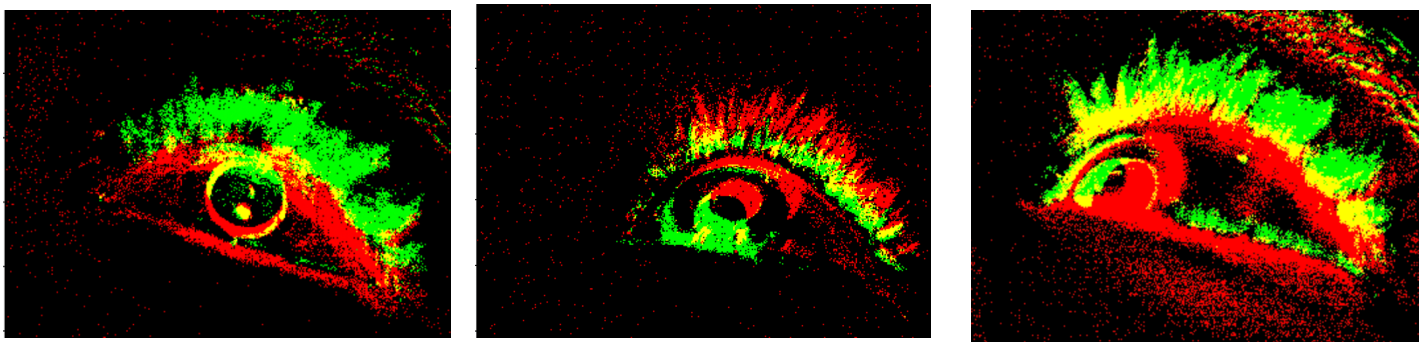

Figure 5.22 Best performing images for 1.1 second time window for stationary with 0.08 (left), slow tracking 0.48 (centre) and fast motion with 0.33 (right) degree errors

5.3.5 Analysis of Issues with Current Methods

After analysis of the current methods in this section, the main problems affecting accuracy for gaze estimation of these methods are as follows:

- **Blinks** - increasing the time window is beneficial but increases the chance of capturing a blink. Blinks not only affect accuracy but reduce sparsity by triggering lots of events.
- **Lack of events in shorter windows -** for stationary motion causing no information on location of eye features.
- **Too many events during fast motion** - when using longer time windows, even without the presence of blinks.

Other discoveries that affect possible power efficiency are:

- **Longer time windows increase the amount of duplicate processing of events** - makes choosing more accurate higher time window come at a cost.
- **Lots of noise** - apparent in all time windows but more pronounced at higher time windows. This reduces sparsity of frames, decreasing power efficiency.
- **More sparsity decreases accuracy in current methods** - to achieve improved accuracy from increased time window, a significant amount of sparsity must be sacrificed.

6 Proposed Improved Representation Method

6.1 Requirements of Solution

Based on the results and analysis in section 5, and the primary issues with current event representations identified in section 5.3.6. The following requirements for an improved method were defined:

- **The method must address blinks** – blinks cause the worst accuracy whilst having a negative impact on sparsity. This was identified as the biggest hurdle to improving both accuracy and power efficiency via more sparse frames.
- **The method must reduce background noise** – noise negatively affects sparsity and potential power efficiency without aiding with accuracy.
- **The method must adjust to the dynamics of the scene** – the previous methods did not adjust to the amount of motion in the scene and as such, there was a trade-off between accuracy in stationary periods and periods of increased motion.
- **The method must use binary valued pixels** – shown to be capable of achieving high accuracy in section 5, this method is aimed for use of low time step, power efficient, low latency SNNs.

- **The method should match baseline accuracy** – accuracy of the current event methods didn't meet that of the standard camera frame baseline, this method shouldn't sacrifice more accuracy for sparsity. Reduced accuracy makes it unlikely to be adopted over existing methods.
- **The model should reduce the average number of non-zero pixels compared to the 0.8 time window** – as seen in the previous section, 0.8 seconds time windows achieved the best accuracy. When looking at the best frame accuracies compared to 0.2 in Figures 5.19 and 5.21, the best accuracy frames of the 0.2 time windows showed similar accuracy to that of the 0.8 window, whilst its average non zero pixels were approximately 15,000 less, showing it is possible to create these frames with a lower number of average non-zero pixels. This method should be capable of creating frames

of the quality seen at 0.2 seconds but more consistently. This will allow for more power efficiency as described in section 3.5, which is the goal of the project set in section 2.

- **The method should have limited additional computational overhead -** when compared to existing approaches. High overhead would undo benefits of any sparsity gains and could make the methods unsuitable for on device AI systems.

6.2 Implementing Solution

6.2.1 Solution for Blinks

To account for blinks, the method used needed to be able to avoid adding blink events to the final frame. This could be done by stopping processing events when a blink is detected, but this would mean in the case when a blink happens at the start of frame creation no events will be captured. Therefore, the events of the blink needed to be skipped, with processing continuing around it. To accomplish this, each time window was split into smaller subsections, see Figure 6.1. For the test set created for the following results, a time window of 1.5 seconds was taken and split into 240 slices, meaning new frames could be produced at a frequency of 160Hz. These slices were then processed separately much like the standard event frames and built into a final frame. Sub windows can be reused for subsequent frames allowing overlapping time periods with limited extra overhead, meaning the theoretical best latency is decided by the time of the sub window.

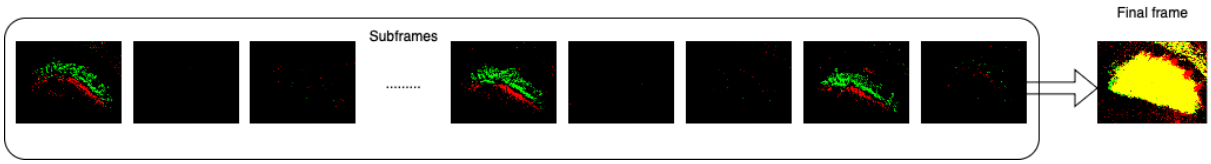

Figure 6.1 Diagram showing how sub windows are stored and built into the final frame

It was noted that periods of blinks were detectable in the sub sections as they would have a much higher number of events compared to any other type of motion. Additionally, it was noted

there would be a greater imbalance in the ratio of positive to negative events when a blink happened within a sub window when compared to other types of motion. This is due to a blink triggering more negative events when the darker eyelid moves down the white of the eye which outweigh the events of covering the pupil, this happens in reverse on the way back up. Any sub windows that contained above a threshold number of events were set to all 0's and the events ignored, see Figure 6.2. In the test set frames, this threshold was set to 900 with smaller threshold at 700, 500, and 300 that would only be included if the ratio of positive to negative events was within a 0.65, 0.7, and 0.85 tolerance respectively. This means the more events there are, the ratio of positive to negative needs to be closer if the sub window is to be included in the final frame.

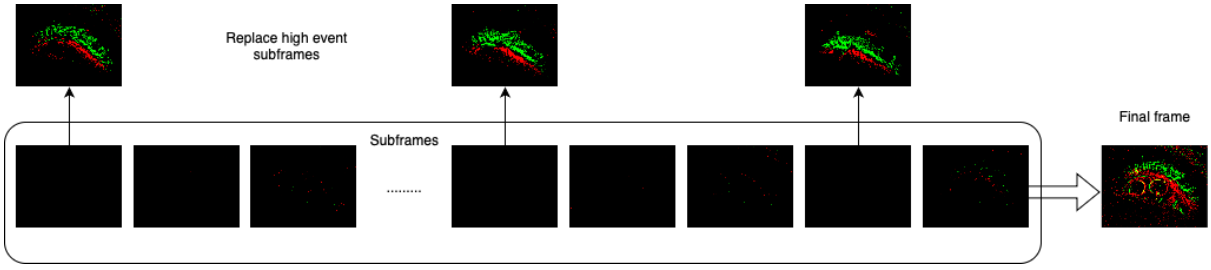

Figure 6.2 Diagram showing how high event sub windows are replaced with blank sub windows to remove blinks.

These thresholds were selected after manual observation of the produced frames looking for values that produced frames consistently like the frames that had the best accuracy in the previous section, but in the presence of blinks.

6.2.2 Reducing Noise

Some noise could also be filtered out during this process by removing sub windows that contained only a small number of events. As there is a somewhat steady amount of noise being generated the number of noisy events in the sub window can be estimated. For the test set any sub windows with less than 40 events were ignored, demonstrated in Figure 6.3.

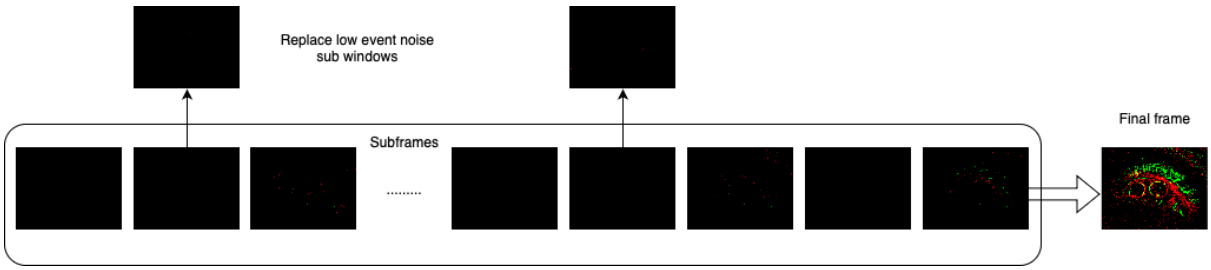

Figure 6.3 Diagram showing removal of noise sub windows.

6.2.3 Dynamic Time Window

Final sub windows were built up into the final frame starting at the newest event window and adding to a final frame, A threshold for total number of events before no further sub windows were added was added, see Figure 6.4. This has the effect of removing the ghosting effect of older events if newer ones are available. This threshold was set at 5000 but decayed over time at a rate of 0.99 times per time window so the required number of events would drop if there were less recent events, meaning even a small amount of movement will stop the frame just gathering noise at some point.

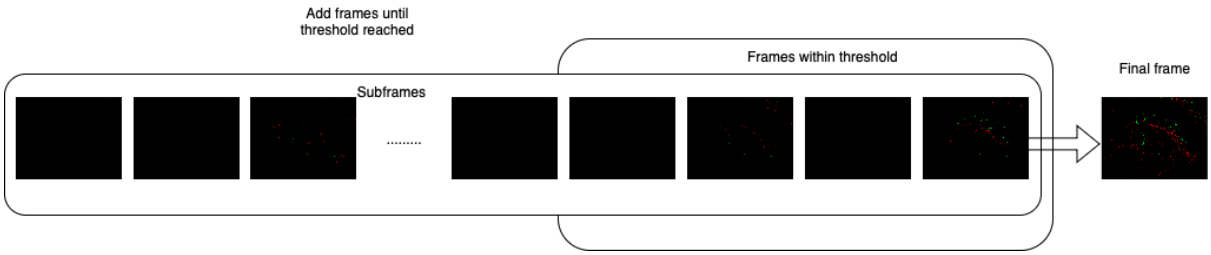

Figure 6.4 Diagram showing how newest sub windows are added until threshold number of events met, resulting only some sub windows being included in the final frame

This created a frame that would dynamically adjust the time window based on the number of events occurring like an adaptive time surface but maintain the special resolution of an event frame whilst also grouping events within a window into sub windows like a voxel representation creating results as shown in Figure 6.5, that had easily identifiable features as shown in Figure 6.6.

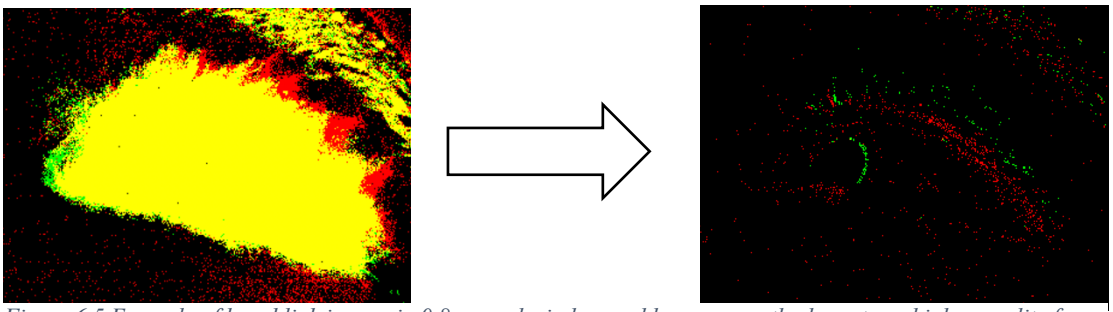

Figure 6.5 Example of how blink is seen in 0.8 second window and how new method creates a higher quality frame

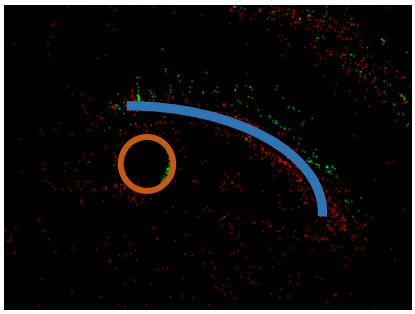

Figure 6.6 Example showing how features of eye are detectable in new frame despite blink. Where ε = the pupil and ς = the eyelid

6.3 Results of New Method

For testing, the final frame was built by setting all pixels to 1 where an event occurred in any of the sub windows, creating a final two-channel image.

	Nonzero pixels	Binary values	Accuracy (degrees)
Traditional Camera	88,723	False	**5.99**
0.2 window	3,746	**True**	10.90
0.8 window	18,630	**True**	6.51
New method	**2,115**	**True**	**6.05**

Figure 6.7 Table showing results of new method against the best of the rest

The results in figure 6.7 show that this method beat the other event methods for accuracy and was only 0.06 degrees worse than the standard camera method. This is with a 98% improvement in sparsity, requiring approximately 2% of the number of spikes compared to the standard camera method if used with an SNN, and being binary so it can be encoded in one

timestep. The new method had even fewer non-zero pixels than the 0.2-time window at 2115 vs 3746.

As can be seen from the worst performing frames and their corresponding accuracy in figure 6.8, the worst performing frames had much less error than the worst performing of the methods in the previous section including the traditional camera at 25.5 and 20.8 vs 45.4 and 36.9, showing that this method is much more consistent at producing good frames for higher precision.

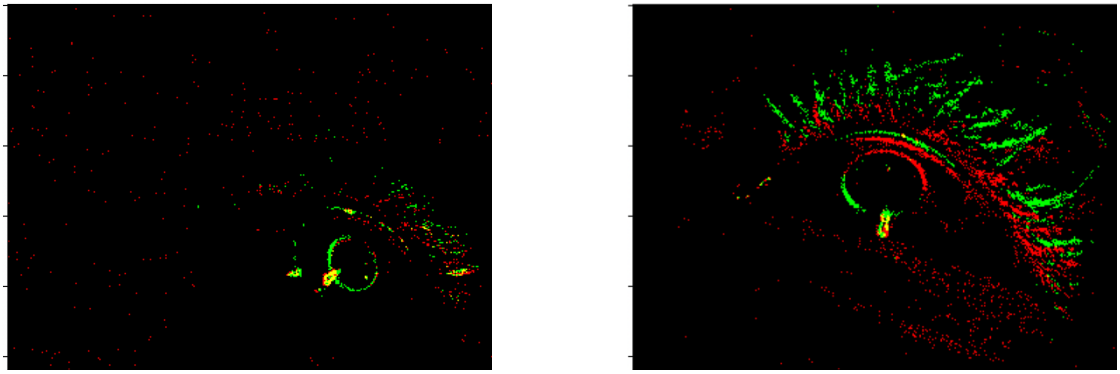

Figure 6.8 Worst accuracy images from the new method with errors of 25.5 (left) and 20.8 (right) degrees

Finally, demonstrated in Figure 6.9, this method can make use of the temporal resolution of the event camera and create good quality frames of the last seen position of eye features even when a blink occurs at the exact timestamp unlike in the standard event frames. This is impossible to do with traditional camera data. As evidenced by the worst-case accuracy being lower for this method when compared to the traditional camera frame model, this could be extremely beneficial and provides a solution to the biggest effect on performance of traditional cameras.

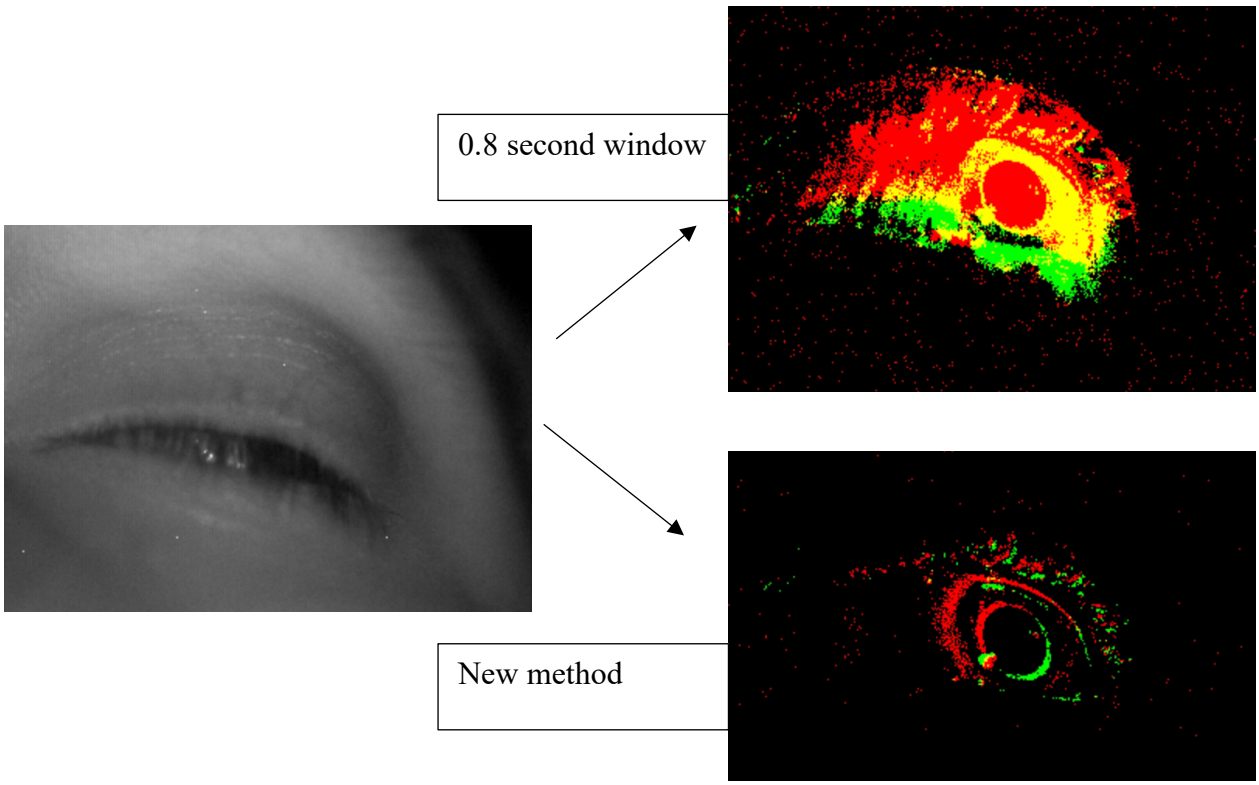

Figure 6.9 Diagram showing difference between effect of blink on 0.8 second time window and new method frame

6.3.1 Applying Further Noise Reduction

The required spikes could be decreased further by applying a simple noise reduction filter before processing. Although normally it is considered better to allow a CNN to handle noise filtering in traditional image processing [135], as this noise filtering will reduce the number of spikes in an SNN it could be worth considering. To filter noise any event in the final frame that has no spatial neighbours in either the positive or negative channel is removed. As shown in figure 6.10, this reduced the average number of spikes from 2115 to 1773, a 16% increase in sparsity, whilst maintaining almost identical accuracy of 6.11 vs 6.05.

	Average number of non-zero pixels	Binary valued	Accuracy (degrees)
Standard	2,115	True	**6.05**
Reduced noise	**1,773**	True	6.11

Figure 6.10 Table showing results of applying noise filtering

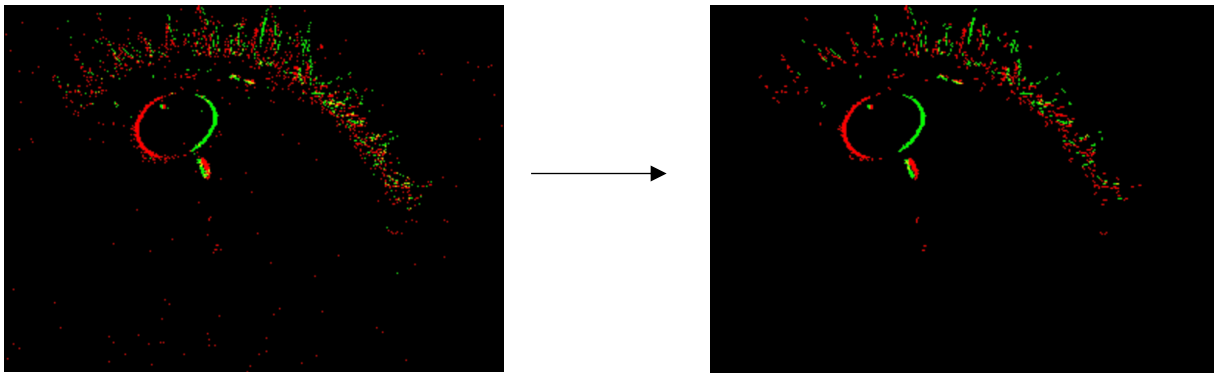

Figure 6.11 Example frame showing original (left) and after noise reduction (right) frames.

This comes at the cost of *2 x W x H x 8* 'or' comparisons to apply, where *W* is the width of the image and *H* the height since the neighbourhood is made up of 8 pixels which must be checked, in each channel, for each pixel. Therefore, it's possible that the spike reduction does not create enough of a power efficiency increase to make it worthwhile.

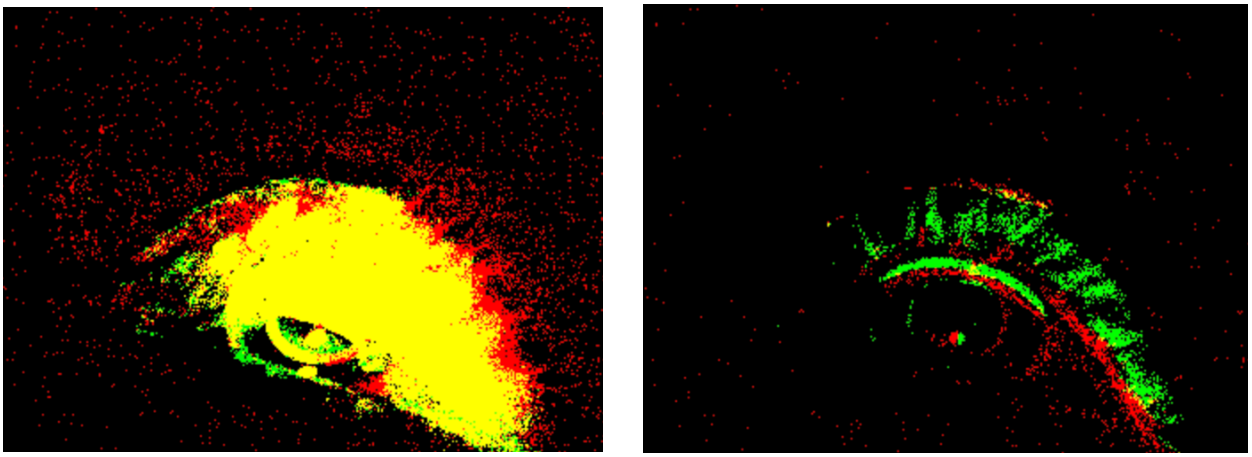

Figure 6.13 Example of blink in a 0.8 second frame (left) and new method (right)

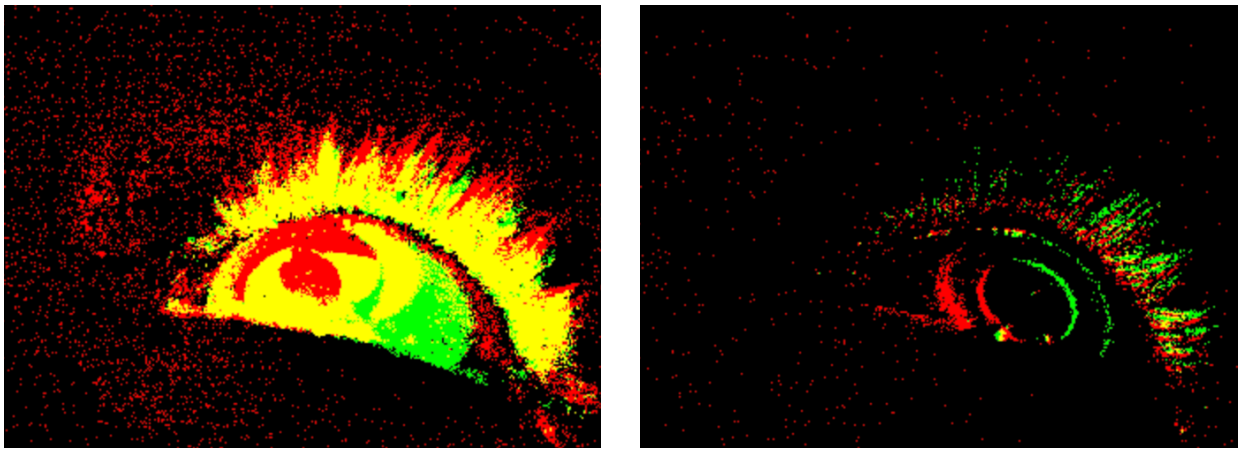

Figure 6.12 Example of fast motion in 0.8 second frame (left) and new method (right)

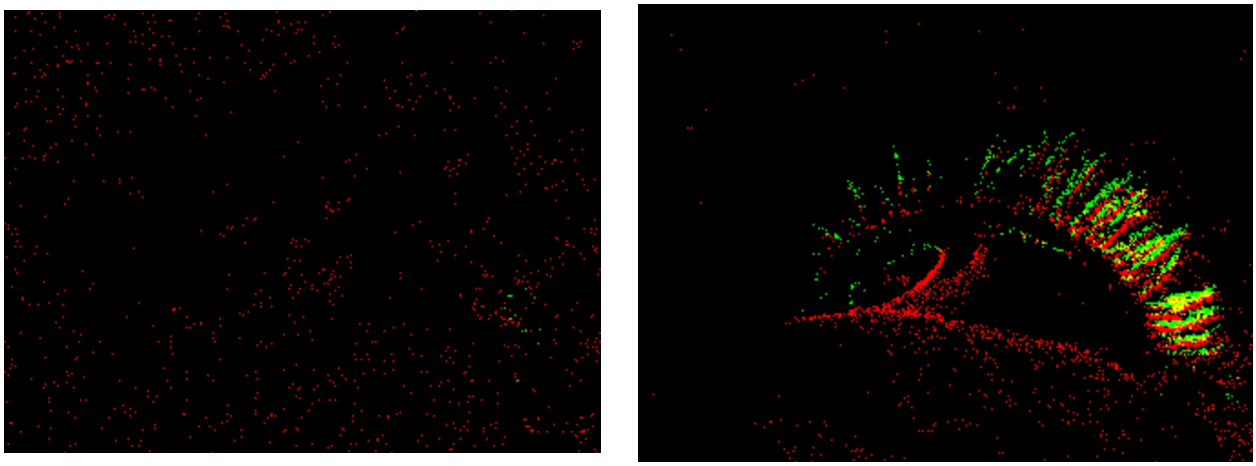

Figure 6.14 Example of stationary eye in 0.5 second window (left) and new method (right)

6.4 Additional Comments

6.4.1 Computational Overhead Analysis

To store a sub window a frame of size H x W will need to be created, where H and W are frame height and width. One of these will need to be created for each sub window, meaning the memory complexity is linear with the number of sub windows. Memory = $H \times W \times SW$, where SW is the number of sub windows. Although for a single frame this is much worse than standard event frames which use one frame. When used for overlapping frames to increase frequency, sub windows can be reused, and frames created from overlapping sub windows, meaning there is no increase in memory requirements. This is not the case for standard event frames, which must then use $H \times W \times OF$, where OF is the number of overlapping frames, for memory.

For runtime, sub window creation is linear with the number of events, like standard event frames for a single frame. However, every pixel in each sub window must be checked when creating the final frame, creating a final runtime of, $(SW \times H \times W) + E$, where E represents the number of events and SW the number of sub windows. For a single frame the runtime of event frames is just E. When using overlapping frames, event frames runtime will be $E \times OF$, where OF is the number of overlapping frames. This means the runtime will be worse, assuming both are running at the same frequency meaning $OF = SW$, as these are indicative of the max frequency.

A rolling frame could be used to reduce the runtime to $E + (2 \times W \times H) + (H \times W)$, where the $(2 \times W \times H)$ is subtracting the oldest sub window and adding the newest to the rolling frame when a new sub window is created, and $(H \times W)$ is clipping every value in the rolling frame to 0 – 1. This is at the cost of an additional frame in memory to store the rolling frame.

This method does potentially come with a computational overhead based on complexity in terms of runtime, or in the case of using the rolling frame, memory. As the number of events processed will be less for the new method the difference is hard to estimate.

6.4.2 Possible Use for Gaze Tracking

This paper focused on single image gaze estimation; however, this method could also easily be modified to be suitable for use with a sequence processing model for gaze tracking. This is outside the capabilities of testing for this paper due to computing hardware restrictions but will be discussed briefly.

Before building the sub window frames into the final frame for processing there is effectively a sequence of tokens that could be passed into a sequence processing model. This could be used without removing the blinks to let the model learn the subframes not to pay much attention to, for example if used with a transformer, or could incorporate the thresholds, or be used in a middle ground where the thresholds are weakened, and the more ambiguous frames are decided by the model.

6.5 Review of Requirements

- **The method must address blinks** – as seen in Figure 6.12, the new method is able to handle blinks and still create a good representation. This is supported by the better worst-case accuracy, which usually contained blinks with existing methods.
- **The method must reduce background noise** – as seen in Figures 6.12, 6.13 and 6.14, there is a noticeable reduction in noise, additionally the method applied in section 6.3.1 can achieve even further noise reduction.
- **The method must adjust to the dynamics of the scene** – as seen in Figures 6.12 and 6.13, the new method can create good frames during high motion, and stationary motion that cause issues for the existing methods.
- **The method must use binary valued pixels** – this is shown in section 6.

- **The method should match baseline accuracy** – as shown in. Figure 6.7, the method got within 0.06 degrees accuracy of baseline and beat existing methods by 0.046. It did not quite beat the baseline but got within a very small margin (<1%).

- **The model should reduce the average number of non-zero pixels compared to the 0.8-time window** – as shown in Figure 6.7, the method reduced average nonzero pixels by 89% when compared to 0.8 second time windows, and even got 44% lower than 0.2 second time windows, without the additional noise reduction.
- **The method should have limited additional computational overhead** – see section 6.4.1. Method unfortunately did have possible additional overhead, although limited, more work needs to be done on analysing this in practice.

7 Limitations and Future Work

This section will discuss limitations of the research in this paper before suggesting future work to improve upon it and resolve some lingering questions.

7.1 Limitations

Limitations of this work can be split into two primary categories, limitations caused by the limited testing on the new method and limitations on the method itself.

The limitations on the testing carried out, which effect the reliability of conclusion that can be made are:

- No SNNs were used for testing. These event camera methods rely on the use of SNNs to achieve the low power, low latency, and high frequency this method intends to be used for. This method of creating event camera representations needs to be tested on an SNN and compared against the existing methods to confirm an increased efficiency. The lack of time to train SNNs and lack of access to neuromorphic hardware to properly analyse an SNNs performance unfortunately made this impossible.
- It's only been used on a single dataset. There is a possibility these results could be an outlier given some characteristic in the dataset meaning they wouldn't generalise, however due to only one dataset being able to be sourced this was unavoidable. The results are also hard to compare as there are limited other works on purely event-based gaze estimation.
- The method was only tested on the specially collected dataset. There's a possibility this didn't capture the intricacies of real eye motion during use of systems that require gaze estimation, for example the dataset is split into 3 types of motions as defined in section 4.3.1, but real data will likely not be this consistent, meaning these results need verification being deployed in a real scenario.

The limitations related to the method developed are:

- The accuracy didn't quite match that of the standard camera baseline, meaning when accuracy is the only consideration this method will not encourage the use of neuromorphic hardware.
- The method comes with an identified additional computational overhead. Although not the case when used in many scenarios when compared against existing methods, more knowledge of the systems this technology aims to be deployed on is required to determine if it fits within specifications.
- The method in its current state uses manually calculated thresholds and does not have a system to change these based on set parameters. This would need to be addressed to allow for more flexibility in the parameters used, e.g., frequency by setting the amount of sub windows.

7.2 Future Work

A lot of the future work will involve more testing to address issues raised in section 7.1. These include:

- Testing the method on an SNN to test if gaze estimation can be achieved using an SNN and whether the developed method improves on the existing approaches.
- Testing of power efficiency gains with this method on an SNN, in particular whether sparser input can be used to achieve less spikes and not only the input layer but at other neurons in models, especially when combined with learning methods that encourage this [93], as discussed in section 3.4.
- Acquiring event cameras to capture and create more data for the method to be tested with would be beneficial. This will also allow for data to be captured during a real scenario, e.g., tracking a user's gaze point whilst they complete everyday tasks on a computer, which will show how it will perform once deployed.

- Improving the method to automatically set thresholds based on parameters, this could also be done using some machine learning to detect characteristics of datasets to aid in finding the optimal thresholds.

Other future works that could be of interest, are applying the same approach to identifying and filtering unwanted events from event representations for other applications. For example, can repeated road markings or pavements be removed from data captured from a car and achieve similar results to those seen here.

Finally, investigation of other model architectures such as capsule networks, discussed in section 3.2, should be carried out to identify whether these can be used to achieve higher accuracy on the task with less parameters. This could be used to inspire SNN models that aim to mimic this performance as seen done with ANNs [78], see section 3.4, to further progress efficiency gains.

8 Conclusion

8.1 Contributions

The main contributions of this paper can be summarised as:

- A review of the state of the art for gaze estimation, efficient deep learning, spiking neural networks and event cameras.
- An evaluation of the suitability of existing methods of using event cameras applied to gaze estimation with a focus on their suitability when combined with SNNs.
- Steps were provided on how an existing event camera dataset can be processed so it can be used for this kind of gaze estimation experiment.
- Existing event processing methods were implemented and tested for gaze estimation.
- Results of existing methods were analysed to see what problems affected performance for gaze estimation.
- The Design and implementation of a new method of processing event data specifically for gaze estimation that achieved a 7% increase in accuracy with an 89% improvement in the sparsity of data.

8.2 Review of Aims

Summaries of the main aims set out for this project and their completionism status are as follows:

- **Investigating event cameras and spiking neural networks as a possible solution to achieve power efficient, low latency, high frequency processing when applied to. Real world task in gaze estimation.** – Methods detailed in section 3, section 4 show how to apply gaze estimation to event camera data. Section 5 shows gaze estimation can be carried out with some accuracy using event camera data. Section 6 presented a method for achieving within 1% of baseline accuracy using only event data, with a 98% increase in sparsity, which based on the review in section 3, is hoped can be used with an SNN to achieve power efficient gaze estimation. Unfortunately, no tests with SNNs

were carried out and as such can't be considered fully completed so this aim was only partially met,

- **A review of the current state of the art in gaze estimation and power efficient AI. –** Section 3 gave a detailed review. This aim is considered met.

- **Investigating the use of sparse data to achieve power efficient gaze estimation –** section 4 details how this can be achieved and section 5 and 6 show it successfully applied. This aim was met based on success of gaze estimation in section 5 with the increase in sparsity of the method in section 6 furthering this.

- **Apply gaze estimation with event camera data and analysis of current methods for processing event data –** Review of current methods were given in section 5. This is considered met.

- **Development of a new method of creating representations to improve accuracy of gaze estimation in a sparser format for use with spiking neural networks to achieve power efficient, low latency, high frequency gaze estimation –** This was achieved in section 6. This is considered met.

8.3 Final Comments

In conclusion this project has successfully met many of the aims set out at the start, whilst providing insights into how the technologies explored in this paper could eventually lead to further advancements in the future, for example discovering it was possible to skip over blinks, the biggest impactor on accuracy. This is something standard cameras cannot do, which could allow for advancements beyond the current state of the art in gaze estimation. It has presented a method to improve upon existing approaches to using event cameras by achieving a 7% increase in accuracy for gaze estimation whilst using 89% sparser data which can be used for more power efficient processing. Additionally, it has shown that comparable accuracy to standard cameras (<1%) can be achieved with a 98% sparser representation, closing the gap between the cutting edge and what can be achieved with efficient neuromorphic hardware.

Hopefully these results will inspire more research into event cameras and spiking neural networks paving the way for this technology to be applied to many other areas, allowing them to reap the benefits of power efficient AI.

9 References

[1] McKinsey, "The state of AI in 2022—and a half decade in review," McKinsey, 6 December 2022. [Online]. Available: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review. [Accessed 26 October 2023].

[2] Z. Zhang and J. Li, "A Review of Artificial Intelligence in Embedded Systems," *Micromachines,* 2023.

[3] R. Singh and S. S. Gill, "Edge AI: A Survey," *Internet of Things and Cyber-Physical Systems,* pp. 71-92, 2023.

[4] Y. Deng, "Deep Learning on Mobile Devices - A Review," *arXiv,* 2019.

[5] M. Martínez-Díaz and Francesc Soriguera, "Autonomous vehicles: theoretical and practical challenges," *Transportation Research Procedia,* vol. 33, 2018.

[6] C. Boretti, P. Bich, F. Pareschi, L. Prono, R. Rovatti and G. Setti, "PEDRo: an Event-based Dataset for Person Detection in Robotics," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2023.

[7] J. L. Zhaoyun Zhang, "A Review of Artificial Intelligence in Embedded Systems," *Micromachines,* vol. 14, no. 897, 2023.

[8] M. S. Bali and S. Khurana, "Effect of latency on network and end user domains in Cloud Computing," *International Conference on Green Computing, Communication and Conservation of Energy,* 2013.

[9] N. Soveizi, F. Turkmen and D. Karastoyanova, "Security and privacy concerns in cloud-based scientific and business workflows: A systematic review,," *Future Generation Computer Systems,,* vol. 148, 2023.

[10] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho and M. Hobbhahn, "Machine Learning Model Sizes and the Parameter Gap," *arXiv,* 2022.

[11] C.-J. Wu and R. Raghavendra, "Sustainable AI: Environmental Implications, Challenges and Opportunities," *ResearchGate,* 2021.

[12] G. R. Yang and X.-J. Wang, "Artificial neural networks for neuroscientists: A primer," *arXiv,* 2020.

[13] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon,* vol. 4, 2018.

[14] S. Schmidgall, J. Achterberg, T. Miconi, L. Kirsch, R. Ziaei, S. P. Hajiseyedrazi and J. Eshraghian, "Brain-inspired learning in artificial neural networks: a review," *arXiv,* 2023.

[15] T.-J. Yang, Y.-H. Chen and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," *arXiv,* 2016.

[16] D. Salvator, "How Sparsity Adds Umph to AI Inference," NVIDIA, 2020. [Online]. Available: https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/. [Accessed October 2023].

[17] Z. Davide, N. Roeland, S. H. Steven and B. S. M., "Sparse Computation in Adaptive Spiking Neural Networks," *Frontiers in Neuroscience,* vol. 12, 2019.

[18] Y. Shang, Y. Li and F. You, "An SNN Construction Method Based on CNN Conversion and Threshold Setting," *Applied Sciences,* 2022.

[19] Fredieu and C. Tanner, "Investigating Continuous Learning in Spiking Neural Networks," *arXiv,* 2023.

[20] E. Lemaire, L. Cordone, A. Castagnetti, P.-E. Novac, J. Courtois and B. Miramond, "An Analytical Estimation of Spiking Neural Networks Energy Efficiency," *arXiv.*

[21] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv,* 2014.

[22] E. Lemaire, L. Cordone, A. Castagnetti, P.-E. Novac, J. Courtois and B. Miramond, "An Analytical Estimation of Spiking Neural Networks Energy Efficiency," *arXiv,* 2022.

[23] PROPHESEE, "MetaVision Sensor," PROPHESEE, [Online]. Available: https://www.prophesee.ai/event-based-sensor-packaged/. [Accessed October 2023].

[24] Inivation, "DAVIS 346," [Online]. Available: https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf. [Accessed October 2023].

[25] U. K. Wolf A, "Contribution of Eye-Tracking to Study Cognitive Impairments Among Clinical Populations," *Front Psychol,* 2021.

[26] N. Valliappan, N. S. Dai and E. e. al, "Accelerating eye movement research via accurate and affordable smartphone eye tracking," 2020.

[27] A. Isayas, M. Paul and f. eelke, "Eye Tracking in Virtual Reality: a Broad Review of Applications and Challenges," *Virtual Reality,* 2023.

[28] Z. G and C. C, "Application of Eye Tracking Technology in Medicine: A Bibliometric Analysis," *PubMed,* 2021.

[29] K.-c. Chen and H. J. Choi, "Visual Attention and Eye Movements".

[30] K. Pfeuffer, J. Obernolte, F. Dietz, V. Mäkelä, L. Sidenmark, P. Manakhov, M. Pakanen and F. Alt, "PalmGazer: Unimanual Eye-hand Menus in Augmented Reality," *arXiv,* 2023.

[31] Apple, "Apple Vision Pro," Apple, [Online]. Available: https://www.apple.com/apple-vision-pro/. [Accessed October 2023].

[32] Corsair, "WHAT IS POLLING RATE?," [Online]. Available: https://www.corsair.com/uk/en/explorer/gamer/mice/what-is-polling-rate-does-it-affect-gaming/#:~:text=The%20polling%20rate%20is%20the,means%201%2C000%20times%20every%20second.. [Accessed October 2023].

[33] Eyelink, "EyeLink 1000 Plus," [Online]. Available: https://www.sr-research.com/eyelink-1000-plus/. [Accessed October 2023].

[34] A. Leube, K. Rifai and K. Rifai, "Sampling rate influences saccade detection in mobile eye tracking of a reading task," *National Library of Medicine,* 2017.

[35] Y. Cheng, H. Wang, Y. Bao and F. Lu, "Appearance-based Gaze Estimation With Deep Learning: A Review and Benchmark," *arXiv,* 2021.

[36] P. Pathirana, S. Senarath, D. Meedeniya and S. Jayarathna, "Eye gaze estimation: A survey on deep learning-based approaches," *Expert Systems with Applications,* vol. 199, 2022.

[37] Y. Lei, S. He, M. Khamis and J. Ye, "An End-to-End Review of Gaze Estimation and its Interactive Applications on Handheld Mobile Devices," *arXiv,* 2023.

[38] A. A. Akinyelu and P. Blignaut, "Convolutional Neural Network-Based Technique for Gaze Estimation on Mobile Devices," *Front Artif Intell,* 2022.

[39] A. Kottwani and A. Kumar, "Eye Gaze Estimation Model Analysis," *arXiv,* 2022.

[40] X. Zhang, Y. Sugano, M. Fritz and A. Bulling, "Appearance-based Gaze Estimation in the Wild (MPIIGaze)," Max Planck Institut, [Online]. Available: https://www.mpi-inf.mpg.de/departments/computer-vision-and-machine-learning/research/gaze-based-human-computer-interaction/appearance-based-gaze-estimation-in-the-wild. [Accessed October 2023].

[41] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik and A. Torralba, "Eye Tracking for Everyone," [Online]. Available: https://gazecapture.csail.mit.edu. [Accessed October 2023].

[42] A. N. Angelopoulos, J. N. Martel, A. P. Kohli, J. Conradt and G. Wetzstein, "EVENT-BASED EYE TRACKING FOR AR/VR | IEEE VR 2021," [Online]. Available: http://www.computationalimaging.org/publications/event-based-eye-tracking/. [Accessed October 2023].

[43] P. Kellnhofer, A. Recasens, S. Stent, W. Matusik and A. Torralba, "Gaze360: Physically Unconstrained Gaze Estimation in the Wild," [Online]. Available: http://gaze360.csail.mit.edu. [Accessed October 2023].

[44] A. Angelopoulos, J. Martel, A. Kohli, J. Conradt and G. Wetzstein, "Event-Based Near-Eye Gaze Tracking Beyond 10,000 Hz.," *IEEE Transactions on Visualization and Computer Graphics,* 2021.

[45] L. Bose, J. Chen, S. J. Carey and P. Dudek, "Pixel Processor Arrays For Low Latency Gaze Estimation," in *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops, VRW 2022*, New Zealand, 2022.

[46] Alzubaidi, Z. L., H. J. and A. e. al, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Big Data,* vol. 8, 2021.

[47] S. Maiti and A. Gupta, "Semi-supervised Contrastive Regression for Estimation of Eye Gaze," *arXiv,* 2023.

[48] A. George and A. Routray, "Real-time Eye Gaze Direction Classification Using Convolutional Neural Network," *International Conference on Signal Processing and Communications (SPCOM),* 2016.

[49] A. A. Abusharha, "Changes in blink rate and ocular symptoms during different reading tasks," *National Library of Medicine,* 2017.

[50] H. KAUR, S. JINDAL and R. MANDUCHI, "Rethinking Model-Based Gaze Estimation," *ACM on Computer Graphics and Interactive Techniques,* vol. 5, no. 2, 2022.

[51] E. Wood and A. Bulling, "EyeTab: Model-based gaze estimation on unmodified tablet computers," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, New York, 2014.

[52] S. SM, S. Z, Z. K, H. A, S. M and P. L. A, "A Driver Gaze Estimation Method Based on Deep Learning," *Sensors,* 2022.

[53] E. T. Wong, S. Yean, Q. Hu, B. S. Lee, J. Liu, R. D. Yean, Q. Hu, B. S. Lee, J. Liu and R. Deepu, "Gaze Estimation Using Residual Neural Network," in *IEEE International Conference on Pervasive Computing and Communications Workshops*, Tokyo, 2019.

[54] L. Huang, Y. Li, X. Wang, H. Wang, A. Bouridane and A. Chaddad, "Gaze Estimation Approach Using Deep Differential Residual Network," *arXiv,* 2022.

[55] S. Ghosh, A. Dhall, M. Hayat, J. Knibbe and Q. Ji, "1 Automatic Gaze Analysis: A Survey of Deep Learning based Approaches," *arXiv,* 2021.

[56] H. Wang, J. O. Oh, H. Jin Chang, J. H. Na, M. Tae, Z. Zhang and S.-I. Choi, "GazeCaps: Gaze Estimation with Self-Attention-Routed Capsules," *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2023.

[57] H. Wang, J. O. Oh, H. Jin Chang, J. H. Na, M. Tae, Z. Zhang and S.-I. Choi, "GazeCaps: Gaze Estimation with Self-Attention-Routed Capsules," *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2023.

[58] U. M. Tomasini, L. Petrini, F. Cagnetta and M. Wyart, "How deep convolutional neural networks lose spatial information with training," *arXiv,* 2022.

[59] C. Palmero, J. Selva, M. A. Bagheri and S. Escalera, "Recurrent CNN for 3D Gaze Estimation using Appearance and Shape Cues," *arXiv,* 2022.

[60] Y. Cheng and F. Lu, "Gaze Estimation using Transformer," *arXiv,* 2021.

[61] J. Liu, S. Tripathi, U. Kurup and M. Shah, "Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey," *arXiv,* 2020.

[62] B. R. Bartoldson, A. S. Morcos, A. Barbu and G. Erlebacher, "The Generalization-Stability Tradeoff In Neural Network Pruning," *arXiv,* 2019.

[63] S. Han, H. Mao and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv,* 2015.

[64] ImageNet, "ImageNet," ImageNet, [Online]. Available: https://www.image-net.org.

[65] M. H. Z. mhzhu and S. Gupta, "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression," *ICLR 2018 Conference Blind Submission,* 2018.

[66] G. MENGHANI, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *arXiv,* 2021.

[67] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *arXiv,* 2021.

[68] D. Liu, X. Chen, C. Ma and X. Liu, "Hyperspherical Quantization: Toward Smaller and More Accurate Models".

[69] J. Pool, "ACCELERATING SPARSITY IN THE NVIDIA AMPERE ARCHITECTURE," Nvidia, [Online]. Available: https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf. [Accessed 25 October 2023].

[70] C. Li, J. Jiang, Y. Zhao, R. Li, E. Wang, X. Zhang and K. Zhao, "Genetic Algorithm based hyper-parameters optimization for transfer Convolutional Neural Network," *arXiv,* 2021.

[71] J. Snoek, H. Larochelle and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *arXiv,* 2012.

[72] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv,* 2015.

[73] G. Chen, W. Choi, X. Yu, T. Han and M. Chandraker, "Learning Efficient Object Detection Models with Knowledge Distillation," 2017.

[74] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk and Q. V. Le, "Rethinking Pre-training and Self-training," *arXiv,* 2020.

[75] C. J. Reed, X. Yue, A. Nrusimha, S. Ebrahimi, V. Vijaykumar, R. Mao, B. Li, S. Zhang, D. Guillory, S. Metzger, K. Keutzer and T. Darrell, "Self-Supervised Pretraining Improves Self-Supervised Pretraining," *arXiv,* 2021.

[76] F. Tan, F. Saleh and B. Martinez, "Effective Self-supervised Pre-training on Low-compute Networks without Distillation," *arXiv,* 2022.

[77] W. Gerstner, W. Kistler, R. Naud and L. Paninski, "Neuronal Dynamics," in *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*, Cambridge, Cambridge University Press, 2014, pp. 3-24.

[78] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong and W. D. Lu, "Training Spiking Neural Networks Using Lessons From Deep Learning," *arXiv,* 2021.

[79] W. Dengyu, Y. Xinping and H. Xiaowei, "A Little Energy Goes a Long Way: Build an Energy-Efficient, Accurate Spiking Neural Network From Convolutional Neural Network," *Frontiers in Neuroscience,* vol. 16, 2022.

[80] A. Yousefzadeh, M. A. Khoei, S. Hosseini, P. Holanda, S. Leroux, O. Moreira, J. Tapson, B. Dhoedt, P. Simoens, T. Serrano-Gotarredona and B. Linares-Barranco, "Asynchronous Spiking Neurons, the natural key to exploit temporal sparsity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 9, 2019.

[81] D. A. Xu, "The High Performance, Low Power Promise of Neuromorphic Computing," Silicon, 20 September 2023. [Online]. Available: https://www.silicon.co.uk/expert-advice/the-high-performance-low-power-promise-of-neuromorphic-computing#:~:text=These%20signals%20accumulate%20inside%20the,%27event%27%2C%20is%20detected.. [Accessed 26 October 26].

[82] N. Rathi and K. Roy, "DIET-SNN: Direct Input Encoding With Leakage and Threshold Optimization in Deep Spiking Neural Networks," *2020,* arXiv.

[83] S. S. Chowdhury, N. Rathi and K. Roy, "ONE TIMESTEP IS ALL YOU NEED: TRAINING SPIKING NEURAL NETWORKS WITH ULTRA LOW LATENCY," *arXiv,* 2021.

[84] G. Datta and P. A. Beerel, "Can Deep Neural Networks be Converted to Ultra Low-Latency Spiking Neural Networks?," *arXiv,* 2021.

[85] Q. Meng, S. Yan, M. Xiao, Y. Wang, Z. Lin and Zhi-Quan Luo, "Training much deeper spiking neural networks with a small number of time-steps," *Neural Networks,* vol. 153, 2022.

[86] M. Dampfhoffer, T. Mesquida, A. Valentian and L. Anghel, "Are SNNs really more energy-efficient than ANNs? An in-depth hardware-aware study," *IEEE Transactions on Emerging Topics in Computational Intelligence,* 2022.

[87] G. Wenzhe, E., F. Mohammed, E. A. M. and S. K. Nabil, "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems," *Frontiers in Neuroscience,* vol. 15, 2021.

[88] S. Oh, S. Lee, S. Y. Woo, D. Kwon, J. Im, J. Hwang, J.-H. Bae, B.-G. Park and J.-H. Lee, "Spiking Neural Networks With Time-to-First-Spike Coding Using TFT-Type Synaptic Device Model," *IEEE Access,* vol. 9, 2021.

[89] Eshraghian and K. Jason, "TUTORIAL 6 - SURROGATE GRADIENT DESCENT IN A CONVOLUTIONAL SNN," snnTorch, [Online]. Available: https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_6.html. [Accessed 22 October 2023].

[90] B. G, S. F, S. A, H. E, S. D, L. R and M. W., "A solution to the learning dilemma for recurrent networks of spiking neuron," *Nat Commun,* 2020.

[91] N. Rathi, G. Srinivasan, P. Panda and K. Roy, "Enabling Deep Spiking Neural Networks with Hybrid Conversion and Spike Timing Dependent Backpropagation," *arXiv,* 2020.

[92] L. J. Haeng, D. Tobi and P. Michael, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience,* vol. 10, 2016.

[93] R. Fontanini, D. Esseni and M. Loghi, "Reducing the Spike Rate in Deep Spiking Neural Networks," 2022.

[94] D. Scaramuzza, "Tutorial on Event-based Cameras:," [Online]. Available: https://rpg.ifi.uzh.ch/docs/scaramuzza/2019.07.11_Scaramuzza_Event_Cameras_Tutorial.pdf. [Accessed 23 October 2023].

[95] Y. a. Z. T. Dong, "Standard and Event Cameras Fusion for Feature Tracking," *ICMVA '21: Proceedings of the 2021 International Conference on Machine Vision and Applications,* 2021.

[96] S. H. Mohamed, MH., J. Heikkonen, H. Tenhunen and J. Plosila, "Towards Real-Time Edge Detection for Event Cameras Based on Lifetime and Dynamic Slicing," *Proceedings of the International Conference on Artificial Intelligence and Computer Vision,* 2020.

[97] P. Felsen, R. Scrofano, R. Rosales, J. Subasavage, N. Desai, T. Smith and M. Dearborn, "Detecting Space Objects in Event Camera Data through 3D Point Cloud Processing," *The Aerospace Corporation.*

[98] P. d. T. D. N. J. M. A. S. Etienne Perot, "Learning to Detect Objects with a 1 Megapixel Event Camera," *arXiv,* 2020.

[99] W. Shariff, M. A. Farooq, J. Lemley and P. Corcoran, "Event-based YOLO Object Detection: Proof of Concept for Forward Perception System," *arXiv,* 2022.

[100] Mingcheng, W. Ziling, Y. Rui, L. Qingjie, X. Shu and T. Huajin, "SCTN: Event-based object tracking with energy-efficient deep convolutional spiking neural networks," *Frontiers in Neuroscience,* vol. 17, 2023.

[101] Z. J, J. S, C. Z, Z. Y and W. Y., "Moving Object Detection and Tracking by Event Frame from Neuromorphic Vision Sensors," *Biomimetics,* 2022.

[102] L. Wang, I. M. Mostafavi, Y.-S. Ho and K.-J. Yoon, "Event-based High Dynamic Range Image and Very High Frame Rate Video Generation using Conditional Generative Adversarial Networks," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.

[103] G. T., L. Y, C. W and e. al., "A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision," 2019.

[104] N. Venkat, "The Curse of Dimensionality: Inside Out," 2018.

[105] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Big Data,* vol. 6, 2019.

[106] M. Abdul Salam, A. Taher, M. Elgendy and K. Mohamed, "The Effect of Different Dimensionality Reduction Techniques on Machine Learning Overfitting Problem," *nternational Journal of Advanced Computer Science and Applications,* vol. 12, 2021.

[107] C. Huang, "Event-based Timestamp Image Encoding Network for Human Action Recognition and Anticipation," 2021.

[108] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce and R. Benosman, "HATS: Histograms of Averaged Time Surfaces for Robust Event-based Object Classification," *arXiv,* 2018.

[109] A. Saeed, H. T. Julia, T. Jonathan, v. S. André and C. Gregory, "Investigation of Event-Based Surfaces for High-Speed Detection, Unsupervised Feature Extraction, and Object Recognition," *Frontiers in Neuroscience,* vol. 12, 2019.

[110] U. M. Nunes, R. Benosman and S.-H. Ieng, "Adaptive Global Decay Process for Event Cameras," 2023.

[111] C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. E. Mahony and D. Scaramuzza, "Fast Image Reconstruction with an Event Camera," *IEEE Winter Conference on Applications of Computer Vision,* 2020.

[112] D. Gehrig, A. Loquercio, K. G. Derpanis and D. Scaramuzza, "End-to-End Learning of Representations for Asynchronous Event-Based Data," *arXiv,* 2019.

[113] A. Z. Zhu, L. Yuan, K. Chaney and K. Daniilidis, "Unsupervised Event-based Optical Flow using Motion Compensation," *arXiv,* 2018.

[114] "event_utils," GitHub, [Online]. Available: https://github.com/TimoStoff/event_utils. [Accessed 23 October 2023].

[115] Z. Li, M. S. Asif and Z. Ma, "Event Transformer," 2022.

[116] Y. Deng, H. Chen, H. Liu and Y. Li, "A Voxel Graph CNN for Object Classification with Event Cameras," *arXiv,* 2022.

[117] Z. S, W. W, L. H and Z. S, "EVtracker: An Event-Driven Spatiotemporal Method for Dynamic Object Tracking," *Sensors,* 2022.

[118] L. Pan, M. Liu and R. Hartley, "Single Image Optical Flow Estimation with an Event Camera," *arXiv,* 2020.

[119] I. L. R., C. Yansong and L. Haizhou, "Is Neuromorphic MNIST Neuromorphic? Analyzing the Discriminative Power of Neuromorphic Datasets in the Time Domain," *Frontiers in Neuroscience,* vol. 15, 2021.

[120] T. Stoffregen, H. D. C. Robinson and A. Fix, "Event-Based Kilohertz Eye Tracking using Coded Differential Lighting," *2022 IEEE/CVF Winter Conference on Applications of Computer Vision,* 2022.

[121] Y. Feng, N. Goulding-Hotta, A. Khan, H. Reyserhove and Y. Zhu, "Real-Time Gaze Tracking with Event-Driven Eye Segmentation," *arXiv,* 2022.

[122] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv,* 2015.

[123] L. Cordone, B. Miramond and P. Thierion, "Object Detection with Spiking Neural Networks on Automotive Event Data," *arXiv,* 2022.

[124] C. Javier, R. Ulysse, C. B. R, B. Francisco and M. Timothée, "Optical flow estimation from event-based cameras and spiking neural networks," *Frontiers in Neuroscience,* vol. 17, 2023.

[125] M. Gehrig, S. B. Shrestha, D. Mouritzen and D. Scaramuzza, "Event-Based Angular Velocity Regression with Spiking Networks," *2020 IEEE International Conference on Robotics and Automation,* 2020.

[126] L. Zhu, X. Wang, Y. Chang, J. Li, T. Huang and Y. Tian, "Event-based Video Reconstruction via Potential-assisted Spiking Neural Network," *IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 2022.

[127] S. Hutton, "Visual Angle," SR Research, [Online]. Available: https://www.sr-research.com/eye-tracking-blog/background/visual-angle/. [Accessed 27 October 2023].

[128] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier and Y. Tian, "Deep Residual Learning in Spiking Neural Networks," *arXiv,* 2021.

[129] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *srXiv,* 2012.

[130] s. kumar, "Building a Gaze Estimator For Full Face Images Using ResNet-50," Medium, September 2021. [Online]. Available: https://srinivas1996kumar.medium.com/building-a-gaze-estimator-for-full-face-images-using-resnet-50-99ffebca313d. [Accessed 21 October 2023].

[131] E. T. Wong, S. Yean, Q. Hu, B. S. Lee, J. Liu and R. Deepu, "Gaze Estimation Using Residual Neural Network," *EEE International Conference on Pervasive Computing and Communications Workshops,* 2019.

[132] Y. Li, J. Chen, J. Ma, X. Wang and W. Zhang, "Gaze Estimation Based on Convolutional Structure and Sliding Window-Based Attention Mechanism," *Sensors,* vol. 13, 2023.

[133] S. M, "Building Resnet-34 model using Pytorch – A Guide for Beginners," 14 September 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/09/building-resnet-34-model-using-pytorch-a-guide-for-beginners/. [Accessed 25 October 2023].

[134] R. Konrad, S. Shrestha and P. Varma, "Near-Eye Display Gaze Tracking via Convolutional Neural Networks," 2016.

[135] D. Sil, A. Dutta and A. Chandra, "Convolutional Neural Networks for Noise Classification and Denoising of Images," *IEEE Region 10 Conference,* 2019.

[136] U. M. Tomasini, L. Petrini, F. Cagnetta and M. Wyart, "How deep convolutional neural networks lose spatial information with training," *arXiv,* 2022.