[redacted]

Department of Electrical Engineering

FINAL YEAR PROJECT REPORT

[redacted]

Background Activity Denoising for Event Camera

Student Name: AU YEUNG Hing Hui, Winnie
[redacted]
Supervisor: Prof BASU, Arindam
Assessor: Prof. JIANG Chaoqiang

Bachelor of Engineering in
Information Engineering

copyright © 2023 all rights reserved

Student Final Year Project Declaration

I have read the student handbook and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I declare that the work submitted for the final year project does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Student Handbook.

Project Title: Background Activity Denoising for Event Camera

Student Name: AU YEUNG Hing Hui [redacted]

[redacted]

Date: 17th December, 2023

No part of this report may be reproduced, stored in a retrieval system, or transcribed in any form or by any means – electronic, mechanical, photocopying, recording or otherwise – without the prior written permission of City University of Hong Kong.

copyright © 2023 all rights reserved

Abstract

Background activity (BA) noise events from Dynamic Vision Sensor (DVS) event cameras are uninformative and grow substantially in low-light conditions. While there are noises, it possibly results in problems including event overload and system instability, which would affect the efficiency and accuracy. Higher noise rate possibly renders existing denoising techniques ineffective. Moreover, comparing algorithm accuracy quantitatively is challenging. This research would measure filtering performance by using known combinations of signal and noise DVS events to better quantify denoising techniques. Three low-cost filtering algorithms are compared using datasets for stationary and moving camera applications of DVS. Algorithm 1 eliminates the majority of the BA noises by using a small fixed size window to measure the time difference between current event and previous events. For more thorough correlation verification, Algorithm 2 improves correlation checking that is proportional to the quantity of pixels. It preserves more signal while removing more noise in comparison to current approaches. To obtain the better accuracy across datasets, Algorithm 3 makes use of a lightweight multilayer perceptron classifier that depends on local event time surfaces and it has overall best performance comparing the previous two algorithms.

Acknowledgements

I would like to express my deep and sincere gratitude to my research supervisor Prof. BASU Arindam and the advisor Mr. SUN Pao-Sheng, for diving me the opportunity to do research and providing invaluable guidance throughout this research. This research could not have been accomplished without the support from Prof. BASU Arindam and Mr. SUN Pao-Sheng.

Contents

List of Figures

1. Introduction

Over the past several years, event cameras have gained traction over the past few years. It is often referred to as a bio-inspired vision sensor that emulates the operation of the human eye and the neurological processing processes involved in vision. Compared to the conventional cameras (traditional cameras), event cameras record a continuous stream of asynchronous event signals rather than a series of signals frame by frame in a sequential manner, giving viewers a real-time picture of how the scene's brightness varies. And noises are unavoidable. There are two major BA noises present in event camera outputs: Leak Noises and Shot Noises.

And these BA noises could cause critical issues including Event Density and False Positives, Calibration and Sensor Calibration, Data Processing and Interpretation, Robustness of Algorithms etc. Without doubt, noises would contribute to the false positive rate. Event cameras' noise might also cause distortions or artifacts in the photos that are taken. This may have an impact on the overall quality of the images, making it difficult to extract accurate and clear visual information from the sceneries. And these images would make the tasks involving tracking and object recognition become more challenging due to the BA noises in event camera outputs. For example, it might make it difficult to accurately estimate object motion or identify objects in complicated situations[3]. Meanwhile, BA noises may affect the sensor calibration. For some tasks, sensor calibration becomes essential to reduce noise-related problems and produce consistent results for ensuring accurate measurements. While processing data, misinterpretation might happen and it overburden the system or some specialized algorithms, which will take BA noise into consideration to prevent incorrect processing or misunderstanding of events. Therefore, how to better avoid the noise production and also effectively filtering noises in the event camera is an essential task. In this research, it focuses on how to better filter the BA noises in event camera outputs and three filtering algorithms would be introduced. Then comparing the filtering performance of these different filtering methods would be completed in this research. The methods would be mentioned in the Section 2 Methodology with details.

1.1 Background

To better filter the BA noises from event cameras, it is essential to understand the characteristics of the BA noises[6]. Thus, event cameras are quite different from conventional cameras by comparing the way they capture signals and also the signal output format. The following section would explain the characteristics of the BA noises from event cameras and also introduce its differences compared to conventional camera output.

1.1.1 The Output of Event Camera

For conventional cameras, it produces sequences of signal frames containing complete pixel information at regular intervals. Unlike conventional cameras, event cameras capture signals by detecting the brightness changes and encode the time. Thus, the output of event cameras is called Events. Event cameras asynchronously output events, which means it generates signal outputs based on local changes in pixel intensity in real time.

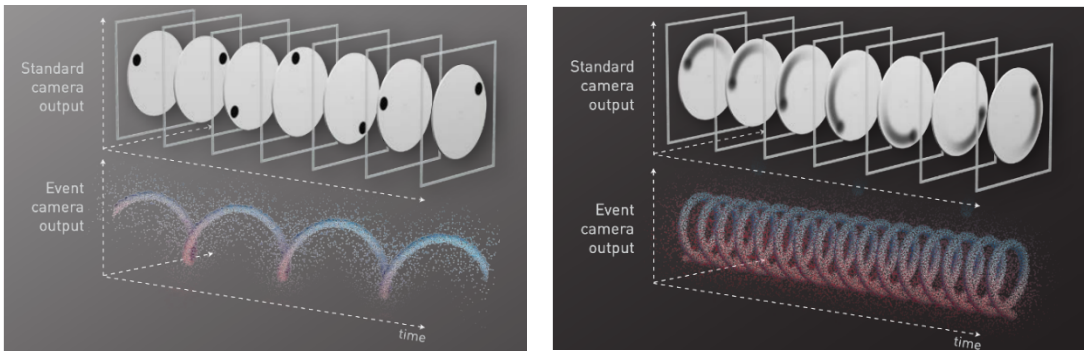

Figure1 The Comparison of Signal Outputs between Event Cameras and Conventional Cameras[4]

Data Format of Each Event[11]:

- Event Coordinates: The X and Y coordinates of the pixel to indicate the location that the event occurred.
- Event Polarity: A binary value (0 or 1) indicating whether the event is ON event or OFF event. The pixel experiencing an increase in intensity is an ON event and a decrease in intensity is an OFF event.
- Event Timestamp: The timestamp is the time while the event occurred and the unit of timestamp is usually in microseconds.

1.1.2 Background Activity Noises in Event Camera

Leak noises and shot noises are two forms of undesired signals that should be filtered out since they might have an impact on the precision and dependability of event-based systems.

1.1.2.1 Leak Noises

Leak noises refers to the occurrence of erroneous events or spurious signals produced by event cameras, even when there are no real changes taking place in the scene. There are several elements that might lead to the generation of leak sounds[9].

a. Sensor Deficiencies: Event cameras, like other electrical devices, may manifest defects in their sensors. The presence of these defects may result in the introduction of several types of noise, including dark current noise, readout noise, and fixed pattern noise. Dark current noise is a result of the inherent thermal energy present in the sensor, while readout noise is attributed to the readout circuitry. Fixed pattern noise may arise due to fluctuations in pixel sensitivity or the non-uniformity of the sensor.

b. Optical Effects: Various optical events, such as lens flare, reflections, or glare, have the potential to induce alterations in the intensity of the camera's field of vision. These visual phenomena have the potential to induce erroneous occurrences, resulting in the generation of spurious acoustic signals. The investigation of optical effects and their influence on event creation is a continuously evolving field of study.

c. Pixel crosstalk refers to the phenomenon when electrical impulses originating from adjacent pixels interact with one another. The presence of interference has the potential to cause erroneous occurrences in neighbouring pixels, hence leading to the occurrence of leakage sounds. Crosstalk may occur as a result of electrical connection or inadequate isolation between pixel sensors.

d. Motion blur may be seen in scenarios when there are things moving at high speeds. Motion blur is a phenomenon that may affect event cameras, resulting in the blurring of filmed

events. The phenomenon of blurring has the potential to generate erroneous occurrences and contribute to the propagation of leakage sounds.

e. Environmental factors may have an impact on the behaviour of event cameras and can lead to the occurrence of false events. These factors include changes in lighting conditions, variations in temperature, and the presence of electromagnetic interference in the surrounding environment. These environmental conditions have the potential to contribute extraneous signals and result in the presence of leakage noise.

f. Event thresholding is a common technique used in event cameras to differentiate between substantial intensity changes and noise. Nevertheless, if the threshold is set too low, there is a risk of erroneously identifying fake events as legitimate events, which might result in the generation of misleading noise.

1.1.2.2 Shot Noises

Shot noises are a manifestation of the intrinsic statistical disturbances that arise in the process of photon arrival[8]. The observed changes in the quantity of photons measured by the sensor may be attributed to the inherent randomness of light. The acoustic characteristics of shot sounds are impacted by many factors:

a. Photon statistics refers to the statistical distribution that governs the arrival of photons, which is often referred to as the Poisson distribution. The occurrence of shot noises may be attributed to the inherent stochasticity in photon detection, leading to variations in the observed event timings and polarity. The variability in these oscillations has the potential to impact the temporal precision and accuracy of event cameras.

b. The Signal-to-Noise Ratio (SNR) is a measure of the relative strength of the desired signal compared to the background noise. In situations with low levels of ambient light or a low SNR, shot sounds become more noticeable and prominent. As the signal-to-noise ratio (SNR)

drops, the inherent variability in photon detection becomes more apparent, resulting in heightened shot noises.

c. Quantum efficiency pertains to the capacity of a sensor to convert incoming photons into electrical signals. An increase in quantum efficiency leads to enhanced signal intensity, hence mitigating the relative influence of shot noises.

1.1.2.3 The Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve is an essential tool in the field of machine learning that is used to assess and visually represent the effectiveness of classification models. The paper offers a thorough examination of the trade-off between sensitivity (also known as the true positive rate) and specificity (the complement of the false positive rate) at various categorization thresholds.

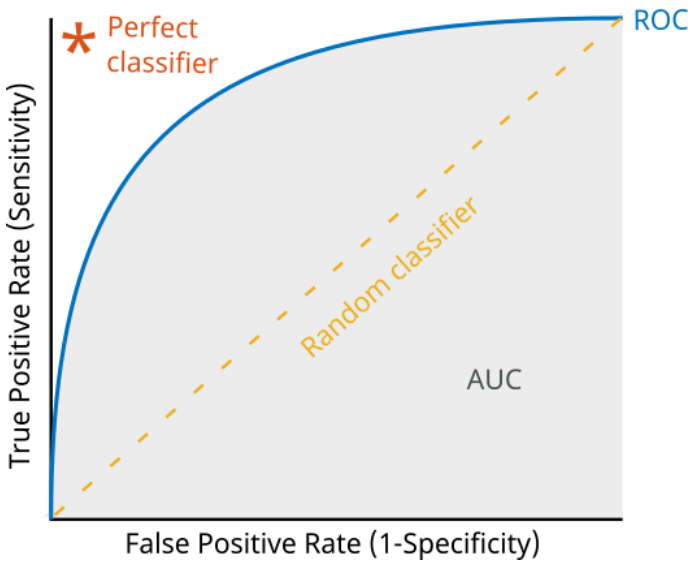

Figure2 The Receiver Operating Characteristic Curve[10]

The ROC curve is used in binary classification scenarios to illustrate the correlation between the true positive rate (TPR) and the false positive rate (FPR) when adjusting the classification threshold. The TPR denotes the ratio of accurately categorized positive occurrences, while FPR quantifies the ratio of erroneously classified negative cases as positive.

To produce a ROC curve, the classification model is first used to provide predictions of probabilities or scores for each individual occurrence within the dataset. Through a systematic manipulation of the classification threshold, the model's predictions are transformed into binary labels. The ROC curve is constructed by calculating the TPR and FPR at various thresholds, yielding a collection of data points[10].

The receiver operating characteristic (ROC) curve offers significant insights into the functioning of the model. A curve that exhibits a near proximity to the upper-left corner of the graph indicates a classifier of high accuracy, characterized by a low incidence of false positives and a high incidence of true positives. In contrast, a curve that exhibits strict adherence to the diagonal line indicates a classifier that is random or ineffectual.

The calculation of the area under the receiver operating characteristic (ROC) curve, often referred to as AUC-ROC, is a significant statistic in the field. The area under the receiver operating characteristic curve (AUC) serves as a measure of the classifier's overall performance, where a larger AUC value signifies a greater degree of discriminative power. A classifier's AUC-ROC value of 0.5 indicates that its performance is comparable to that of random guessing, while an AUC-ROC value of 1.0 suggests a classifier that is flawless in its predictions.

The receiver operating characteristic (ROC) curve and the area under the ROC curve (AUC-ROC) are very valuable tools in the context of model selection and comparison. When assessing the performance of different classifiers, the curve offers a graphical depiction of their comparative effectiveness. Moreover, the area under the receiver operating characteristic curve (AUC-ROC) enables a quantitative evaluation and ranking of classifiers according to their capacity to discriminate.

In essence, the receiver operating characteristic (ROC) curve serves as a visual depiction of the balance between the rate of correctly identified positive instances and the rate of incorrectly identified negative instances, as it varies with various thresholds for classification.

The use of this technique proves to be very effective in the assessment and comparison of classification models' performance. Through the examination of the shape of the curve and the computation of the region under it, professionals can evaluate the precision and discriminatory capability of their models, so facilitating well-informed decision-making in the realm of machine learning endeavors.

1.1.3 Relevant Work

1.1.3.1 Spatiotemporal Correlation method to Filter Noises

The Background Activity Filter (BAF) is the predominant approach for denoising BA in DVS, as reported by previous studies[7]. The BAF determines that an event is classified as a 'signal' if any previous event has taken place inside the spatiotemporal volume defined by the immediate closest neighbors in space and the requirement that the timestamp of the nearest neighbor(NNb) event, denoted as 't_{NNb}', is less than the timestamp of the current event, denoted as 't_e', with respect to the correlation time threshold, denoted as 't'. If the specified criteria $t_{NNb} - t_e < T$ is satisfied, the occurrence is categorized as a signal event. Alternatively, if the event does not meet the criteria for classification, it is deemed a noise occurrence and thereafter disregarded.

Spatiotemporal correlation method depicts the relationship between the event timestamp (t_e) and the timestamps (t_{NNb}) associated with the NNb pixels at coordinates (x, y)[11]. These coordinates are determined by the constraints $|x_e - x| \leq 1$ and $|y_e - y| \leq 1$, where (x_e, y_e) represents the address of the e event. The t_{NNb} are derived from a two-dimensional Timestamp Image (TI) that represents the most recent event timestamps recorded at each pixel. Given that the normal range for the parameter t is often established within the interval of 1 to 100 milliseconds, it is plausible to consider quantizing the Time Interval (TI) to a resolution of 1 millisecond.

a. Increasing Neighborhood by Subsampling

To enhance the efficiency of event storage, the x and y event addresses are subjected to a right-shift operation by b bits prior to their use. This enables each TI pixel to retain the most

recent event inside a 2b x 2b DVS pixel block[11]. The process of subsampling results in a reduction in TI memory by a factor of 2b. This technique allows weakly correlated signals to travel across long distances without significant computational burden on the filter.

b. Hot Pixel Filtering by Self-Exclusion

A pixel that is often referred to as a "hot pixel" has an unusually high occurrence of noise occurrences[11]. The occurrence of these discrepancies may be attributed to the mismatch between transistors and devices. Hot pixel events, while they should ideally be excluded by the filter, may constitute a significant portion of the output in situations with high levels of illumination. If the pixel itself is considered part of the neighborhood, hot pixels will transmit their events if they exceed the threshold correlation time. Through the exclusion of self-correlation, the BAF method inherently eliminates such correlations.

1.2 Objectives

- Implement Background Activity Filter with Different Correlation Time
- Implement Spatiotemporal Correlation Filter with Different K and Different Correlation Time
- Build a Classifier with Neural Network
- Implement Multilayer Perceptron Denoising Filter
- Compare the Performances of Three Filters
- Visualize the Filtering Performances

2. Methodology

2.1 Create Testing Datasets

There are two datasets would be used in this research. The first one is “Hotel-Bar” dataset which is from stationary camera. And another one is “Driving” dataset from moving camera. Thus, these two datasets “Hotel-bar” and “Driving”, both are captured by DAVIS 346 event camera. And labelled leak and shot noises were added into these clean datasets respectively by using jEAR software toolkit. There are four testing datasets created and used in this research.

2.1.1 Stationary Camera and Moving Camera

The use of event cameras in stationary camera settings pertains to situations when the camera is positioned in a fixed and immobile state. This application is particularly well-suited for the purposes of monitoring and surveillance. Event cameras have superior performance in recording dynamic events and detecting changes in the visual environment, exhibiting remarkable temporal resolution and minimal latency. By using an event-driven approach, the system selectively transmits data just in response to substantial changes, so effectively preserving bandwidth and storage resources. Stationary event cameras have the capability to effectively collect and transmit events, including but not limited to object movements, human actions, or abnormalities. This characteristic renders them highly suitable for deployment in many domains such as security systems, traffic monitoring, and environmental monitoring.

Conversely, the use of event cameras in the context of moving camera applications pertains to instances whereby the camera undergoes motion[13]. Event cameras provide significant benefits in recording high-speed motion and dynamic scenes, mostly attributed to their asynchronous event capture capability and low latency. These qualities render them well-suited for several applications, including autonomous cars, robotics, and action recognition. The cameras include an event-driven characteristic that enables them to accurately record temporal data and effectively monitor objects in motion. This capability offers significant visual information that may be used for immediate decision-making and control systems. The

lightweight and low-power attributes of these devices render them well-suited for applications in the mobile and wearable domains.

2.1.2 DAVIS 346 Event Camera

The DAVIS 346 event camera is an innovative and proficient event-based vision sensor created by iniLabs, a prominent supplier of event-based sensing systems. The DAVIS (Dynamic and Active-pixel Vision Sensor) series signifies a notable progression within the domain of event cameras.

Figure3 The DAVIS 346 Event Camera[2]

The DAVIS 346 event camera is equipped with a sensor that has a resolution of 346x260 pixels. This sensor functions in an event-driven way, allowing it to capture changes and motion in the scene with exceptional speed and efficiency. The device has a high temporal resolution in the order of microseconds, allowing for the precise capturing of fast-paced events and dynamic sceneries.

The DAVIS 346 has a notable characteristic in the form of reduced latency[1], which enables the provision of real-time visual information without necessitating frame-to-frame processing. This characteristic renders it well-suited for use in applications that need prompt and time-critical reactions, such as robots, autonomous cars, and augmented reality.

2.1.3 jEAR - Software Toolkit

The jEAR is a powerful Java software toolkit created by the Sensors Group, Inst. Of Neuroinformatics, UZH-ETH Zurich, specifically designed for Address-Event Representation (AER) neuromorphic vision and audio sensor processing. Developed to facilitate research and

development in the field of neuromorphic computing, jEAR provides a comprehensive set of tools for analysing, visualizing, and processing data from AER-based vision and audio sensors. With jEAR, researchers and developers can efficiently handle AER data streams, implement complex processing algorithms, and leverage the capabilities of neuromorphic systems. jEAR's user-friendly interface and extensive library of functions make it an invaluable resource for anyone working with AER-based neuromorphic sensors, enabling them to explore and exploit the potential of these advanced sensor technologies in various applications such as robotics, artificial intelligence, and sensory processing.

2.1.4 Create Labelled Noises

A software toolkit named “jEAR” was used for adding leak noises and shot noises into the clean datasets “Hotel-bar” and “Driving”. As the figure3 shown, the clean dataset “Hotel-bar” loaded into the jEAR software toolkit.

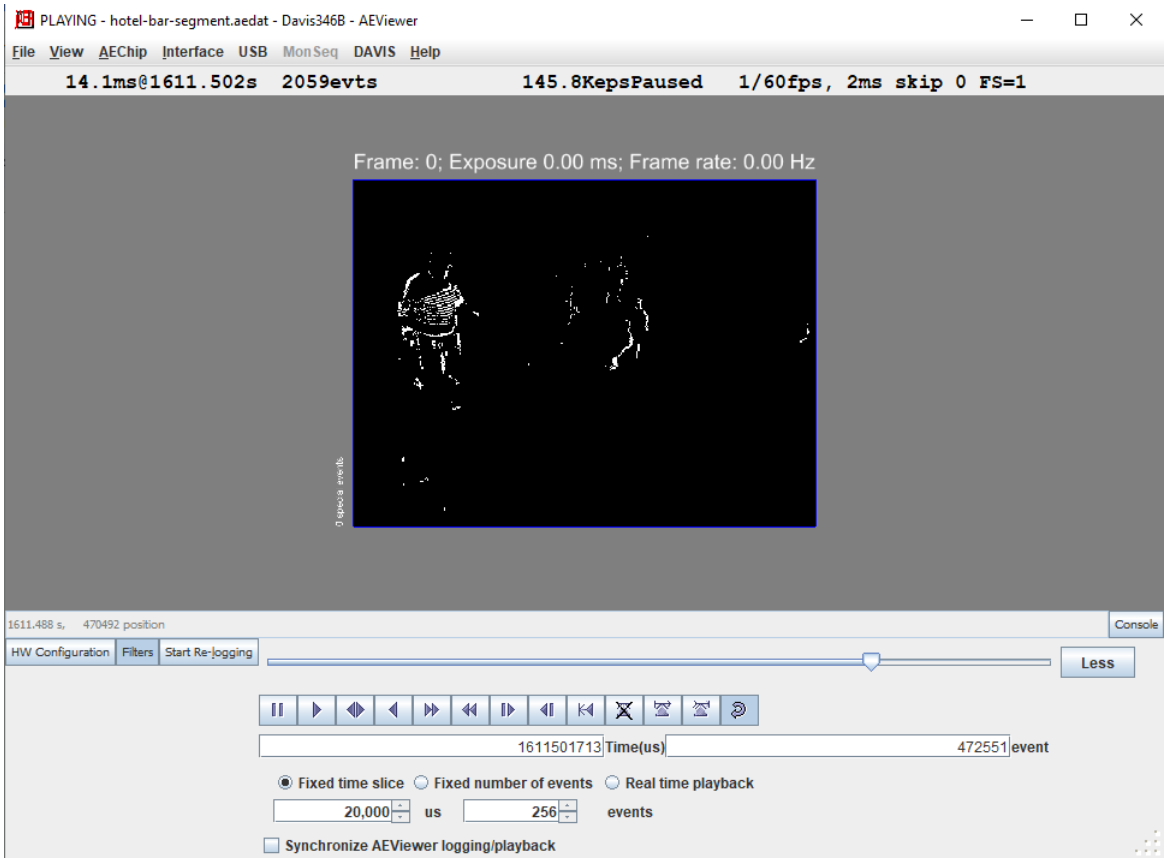

Figure4 The Clean Hotel-bar Dataset Loaded into the jEAR

Then, under the “Filters” tab, there is a noise control function that allow users to adjust the leak noises and shot noises and it would add noises into the datasets based on the parameter users set up.

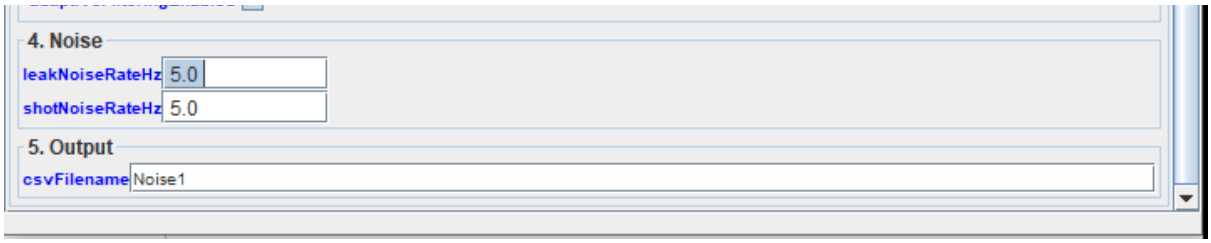

Figure5 The Noise Control Function in jEAR

In this research, the leak noise rate is 5.0 Hz and the shot noise rate is 5.0 Hz as well. The following image shows four testing datasets generated by jEAR software after adding the leak noises and shot noises respectively.

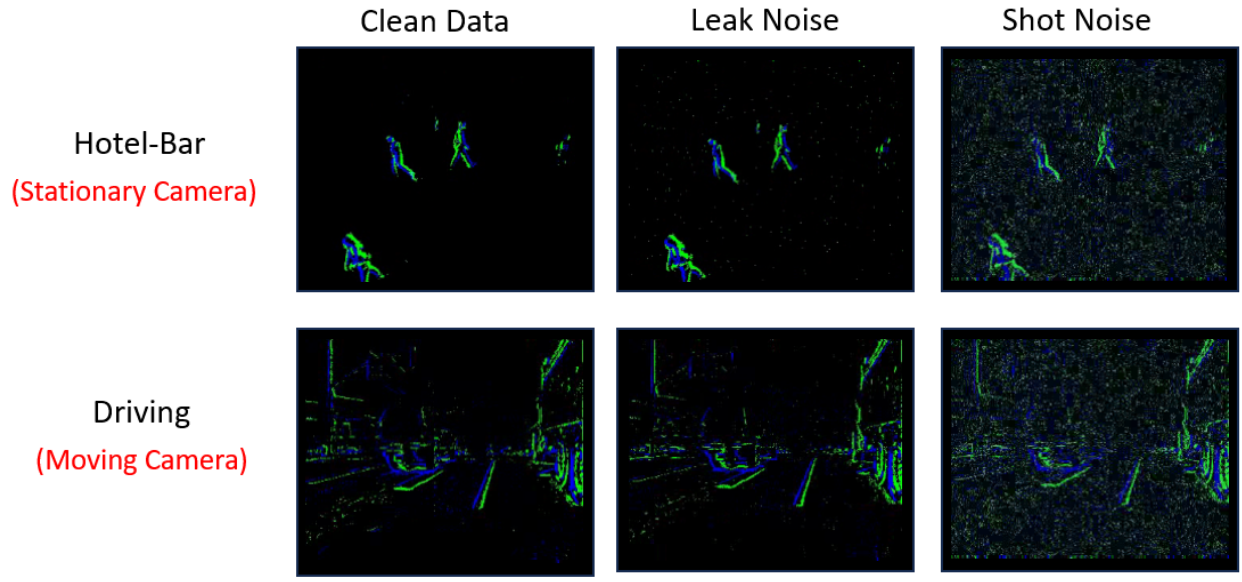

Figure6 The Four Testing Datasets Generated by jEAR

2.2 The Background Activity Filter (BAF)

2.2.1 Algorithm Explanation

The purpose of BAF is rooted in the nature of event cameras, which produce asynchronous, event-driven data streams[5]. The background activity filter (BAF) is an essential element in the operation of event cameras, specifically designed to effectively handle visual input. The BAF functions by analysing the behavioural patterns of adjacent events and deciding as to whether the present event may be categorized as a signal or as a component of the background activity. The successful differentiation of significant events from noise is achieved by the BAF via the use of the concepts of time differences and correlation time.

Upon the occurrence of an event inside the camera's visual range, the Behaviour Analysis Framework commences its analysis by considering the temporal dimensions of the event. The primary emphasis is on the event's nearest neighbours, which refer to the events that are geographically in closest proximity to it. Through the analysis of these adjacent occurrences, the BAF seeks to discern patterns and associations within the temporal realm.

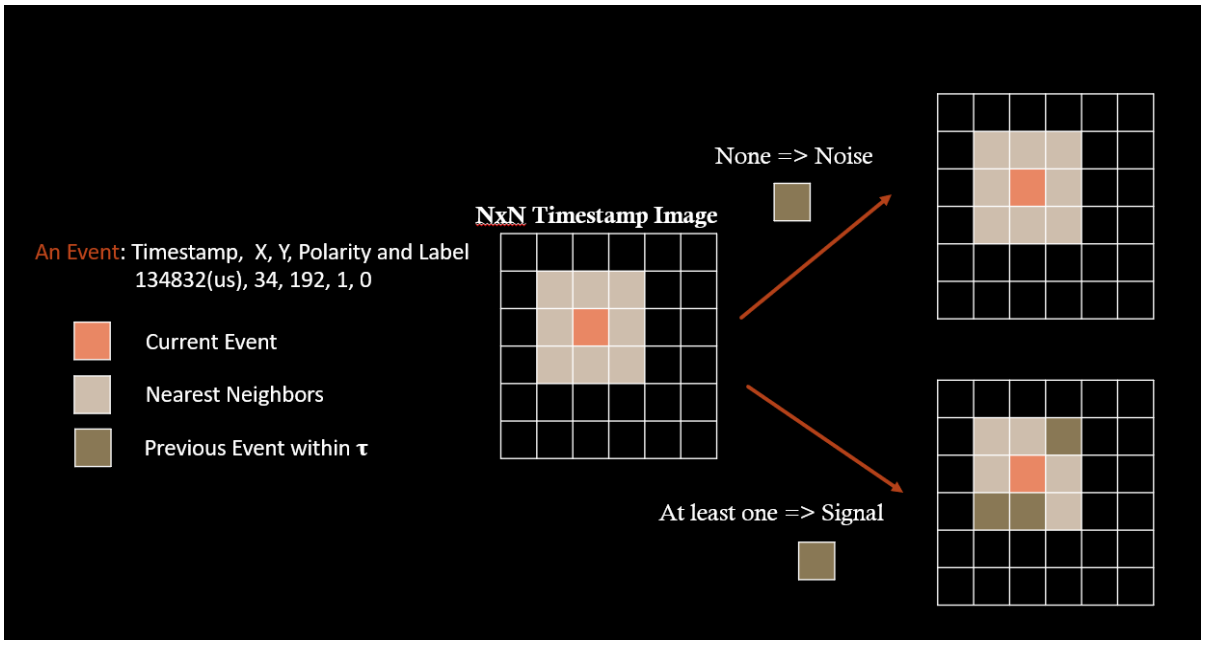

Figure7 The Background Activity Filter (BAF)

The BAF algorithm computes the temporal disparity between the present occurrence and its closest neighbouring events. The temporal gap between the occurrence of the current event and its nearby occurrences is represented by this time difference. This metric plays a pivotal role in assessing the importance of the present occurrence.

In order to determine the classification of the present occurrence as a signal, the BAF conducts a comparison between the computed temporal disparity and a pre-established correlation duration, indicated as τ. The correlation time functions as a criterion, establishing the upper limit of the temporal interval during which an event may be classified as part of the background activity. The event is classified as a signal by the BAF if the temporal gap between it and its closest neighbouring events is less than the threshold value τ.

Through the use of this process, the BAF efficiently eliminates extraneous information and detects occurrences that exhibit temporal correlation, therefore differentiating them as significant signals. This procedure enables event cameras to prioritize pertinent information, hence improving their efficacy in acquiring and processing visual data.

The use of the background activity filter has shown its significant utility across a range of applications. The Background Activity Filter (BAF) allows event cameras to selectively attend to major changes or items of interest in dynamic settings characterized by frequent occurrences, while effectively filtering out extraneous background activity. The method of selectively processing information serves to decrease the computational burden and enhance the overall performance of the camera.

2.2.2 Implementation

In this research, it manipulated the correlation time as an independent variable and assesses the efficacy of the filter via the generation of a ROC curve. The shown curve depicts the relationship between the true positive rate and the false positive rate over various correlation time thresholds. This allows for an evaluation of the efficacy of the BAF in discerning signal from noise.

2.3 The Spatiotemporal Correlation Filter (STCP)

2.3.1 Algorithm Explanation

The Spatiotemporal Correlation Filter (STCF) is a crucial filter used in event cameras for the purpose of examining and categorizing occurrences according to their spatiotemporal attributes. The STCF and the Background Activity Filter (BAF) have a same operational principle, which involves analysing the temporal intervals between the present event and its closest neighbouring events. In contrast to the BAF, which primarily focuses on the actions of individual neighbours, the STCF takes into account the combined behaviour of several neighbours in order to arrive at its conclusion.

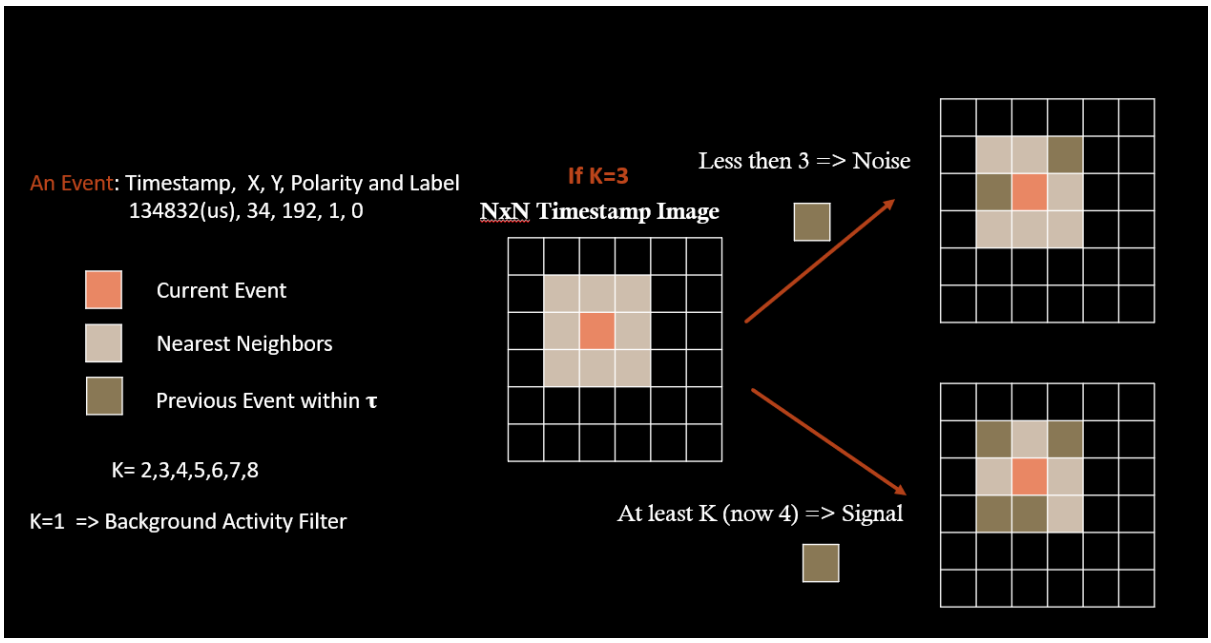

Figure8 The Spatiotemporal Correlation Filter (STCP)

When an event takes place within the visual range of the event camera, the STCF is activated to assess the temporal connections between the event and its closest neighbouring events. The algorithm takes into account the temporal disparities between the present event and its adjacent events, effectively recording the temporal intervals between their happenings.

In the case of the STCF, a parameter known as K is added, denoting the needed number of neighbours with temporal differences smaller than the correlation time τ. The correlation time functions as a criterion, establishing the upper limit of the temporal interval during which an event may be classified as part of the background activity.

In order to categorize the present occurrence, the STCF assesses the quantity of its closest neighbouring instances that exhibit temporal disparities less than the threshold value denoted as τ [12]. If the quantity of neighbouring instances satisfying this condition is equal to or over the threshold value K, the present occurrence is classified as a signal. On the other hand, if the count of neighbouring events with time differences less than τ is below the threshold K, the present event is categorized as noise.

The STCF utilizes this methodology to harness the combined behaviour of adjacent events in order to determine the importance of the present event. This approach facilitates the filter's enhanced ability to distinguish between significant signals and extraneous background noise.

The STCF filter has several benefits when used in event camera applications. By including the temporal correlations of numerous neighbouring data points, the filter offers a more resilient evaluation of the present occurrence. This strategy helps eliminate false positives and increases the accuracy of event categorization.

Moreover, the capacity of the STCF to capture spatiotemporal correlations enhances the comprehension of the dynamics within the scene. The filter facilitates the detection of intricate patterns, object tracking, and identification of significant changes within the visual environment via the examination of the collective behaviour of nearby events.

To summarize, STCF is a filtering mechanism used in event cameras for assessing the importance of events by considering the temporal connections between the present event and its closest neighbouring occurrences. The effectiveness of the STCF in differentiating signals from noise is achieved by the evaluation of the count of neighbouring elements with temporal

differences that are below a specified correlation time τ. The exploitation of spatiotemporal correlations by this filter improves the accuracy of event categorization and offers vital insights into the dynamics of the seen scene.

2.3.2 Implementation

The Spatiotemporal Correlation Filter (STCF) is the second filter in the study that takes into account the temporal disparity between the present event and its closest neighbouring events. A criteria was introduced that is based on the count of neighbouring elements exhibiting a temporal separation less than the correlation time. The study assessed the efficacy of filtering performance via the examination of various correlation periods and values of K. The data obtained from the experiment was organized in a tabular format, which allows for a systematic comparison of the filtering performance under various parameter configurations.

2.4 The Multilayer Perceptron Denoising Filter (MLPF)

2.4.1 Algorithm Explanation

Due to their reliance on counting recent events in the nearest neighbour buffer (NNb), correlation-based denoising algorithms are limited in their ability to identify spatiotemporal structural cues that are valuable for distinguishing between signal and noise events. In order to investigate the potential improvement in denoising accuracy, we have devised a Deep Neural Network (DNN) denoiser using a basic Multilayer Perceptron (MLP) architecture[11]. The objective is to assess the performance of a lightweight classifier trained on annotated data.

The Figure8 illustrates the implementation of our Multi-Layer Perceptron Framework (MLPF), which incorporates a solitary hidden layer consisting of NMLPF neurons. These neurons are stimulated by a cluster of S^2 MLPF pixels originating from the Temporal Information (TI) of the Neural Network b (NNb) in the vicinity of the target event that requires classification. In order to ascertain the distinction between signal and noise in an event, it is crucial to possess knowledge on the temporal proximity of surrounding

occurrences. Recent events have significance in this determination, whereas older events may be deemed inconsequential and therefore dismissed. The Ax,y input channel is responsible for encoding the age of NNb events in the form of a Time Surface (TS) . The value of Ax,y is determined for each pixel in the TI by using the equation shown in Figure8 .

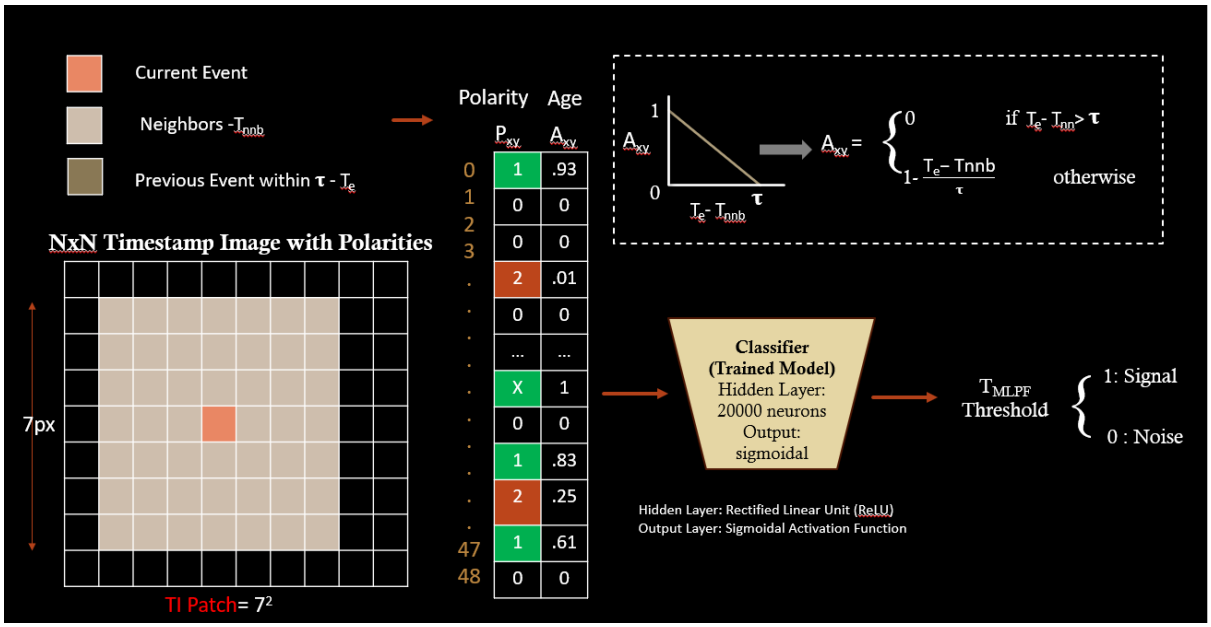

Figure9 The Multilayer Perceptron Denoising Filter (MLPF)

The MLPF is a filtering methodology designed to discern between signal and noise in event-based data. This conclusion is made by considering the polarity and temporal disparity of adjacent occurrences.

At the beginning, MLPF establishes the concept of a patch encompassing every individual pixel, so establishing the criteria for determining the quantity of adjacent events to be taken into account. Each pixel inside the patch has two distinct attributes: Polarity(Px,y) and Age(Ax,y). The polarity refers to the categorization of an event as either positive or negative, while the Age denotes the temporal gap between the current event and its adjacent events.

The Age data are often normalized to a range of 0 to 1 by a formula that is explicitly defined in the Figure8. The method of scaling enables the consistent comparison and processing of temporal disparity data.

After acquiring the patch values, they are then entered into a classifier. The classifier is a model that has undergone training and acquired the ability to distinguish between signal and noise by using the provided input characteristics. The classifier algorithm analyses the patch values and generates an output value within the range of 0 to 1.

The subsequent stage is doing a comparison between the output of the classifier and a pre-established threshold. The aforementioned threshold functions as a criteria by which the classification of an ongoing occurrence as either signal or noise is determined. In the case that the output of the classifier over the predetermined threshold, it is deemed a signal. Conversely, in the case that the output value is lower than the predetermined threshold, the occurrence is categorized as noise.

The objective of using the MLPF approach is to efficiently eliminate extraneous noise while preserving significant signal events within the data. The classifier assumes a pivotal role in determining the classification outcome, leveraging key information such as polarity and temporal disparity.

It is crucial to acknowledge that the particularities of MLPF, such as the dimensions of the patch, the scaling formula, the architecture of the classifier, and the threshold, may differ based on the particular implementation and demands of the application or system.

To summarize, the Multilayer Perceptron Denoising Filter utilizes adjacent event data, including polarity and temporal disparities, in order to distinguish between signal and noise. The use of patch values in a trained classifier, followed by a comparison of its output against a predetermined threshold, enables the MLPF to effectively detect and preserve significant events while eliminating irrelevant noise from event-based datasets.

2.4.2 Classifier Training

2.4.2.1 TensorFlow Keras - API

TensorFlow Keras is a very robust and accessible deep learning framework that is extensively used for the purpose of training artificial intelligence (AI) models[14]. It offers a high-level interface for creating, training, and deploying neural networks, making it accessible to both novices and seasoned practitioners in the area of AI.

Keras, which is seamlessly integrated into the TensorFlow environment, provides a user-friendly and simple approach for designing and training deep learning models. The simplicity and abstraction of this approach allow developers to direct their attention on the model design and problem-solving elements, rather than being burdened by the intricacies of low-level implementation details.

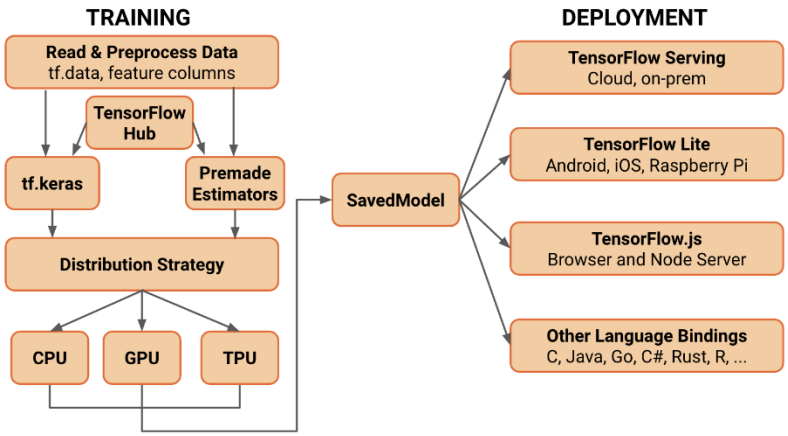

Figure10 The Conceptual Diagram of TensorFlow Keras[15]

The modular and flexible design of TensorFlow Keras is considered to be one of its prominent characteristics. This software platform facilitates the construction of neural networks by the sequential arrangement of layers, hence simplifying the process of designing intricate models that include diverse layer types, including convolutional, recurrent, and dense layers. In addition to its core functionality, Keras offers a diverse selection of pre-existing layers and activation functions that may be readily tailored to meet unique needs.

The process of training artificial intelligence (AI) models using the TensorFlow framework. Keras is considered to be simple because to its interface that is designed to be easily used by users. The process involves developers defining the architectural structure of the model, then compiling it by specifying a loss function and optimizer, and ultimately training it using labelled data. Moreover, Keras offers a comprehensive set of evaluation metrics that allow for the monitoring of the model's performance throughout the training process. This functionality enables users to evaluate the correctness of the model and make any required modifications accordingly.

One noteworthy characteristic of TensorFlow Keras is its ability to function well on many hardware platforms, including central processing units (CPUs), graphics processing units (GPUs), and specialized hardware such as Tensor Processing Units (TPUs)[15]. This feature enables effective and expedited training, making it applicable to a diverse array of contexts.

In addition, TensorFlow Keras exhibits a smooth integration with several other components of TensorFlow, including TensorFlow Datasets and TensorFlow Hub. This integration facilitates the simplification of both data preparation and model deployment procedures. This interface facilitates users in conveniently accessing and using pre-existing datasets and pre-trained models, hence reducing the time and effort required for the development of artificial intelligence solutions.

In summary, TensorFlow Keras provides a robust and adaptable framework for the training of artificial intelligence models. The popularity of this software stems from its user-friendly interface, modular design, and ability to function on several hardware platforms, which appeals to both academics and practitioners. TensorFlow Keras offers a simplified workflow for individuals at all levels of expertise in the area of artificial intelligence, enabling them to build, train, and deploy deep learning models. This framework empowers users to engage in creative and innovative pursuits within the realm of AI.

2.4.2.2 Rectified Linear Unit Activation Function

The Rectified Linear Unit (ReLU) activation function is widely used in the TensorFlow Keras API for the purpose of model training. The activation function discussed above is a simple but efficient mechanism that incorporates non-linear characteristics into neural networks. The rectified linear unit (ReLU) function has been widely embraced in the field of machine learning owing to its effectiveness in addressing the vanishing gradient issue and enhancing the efficiency of model training.

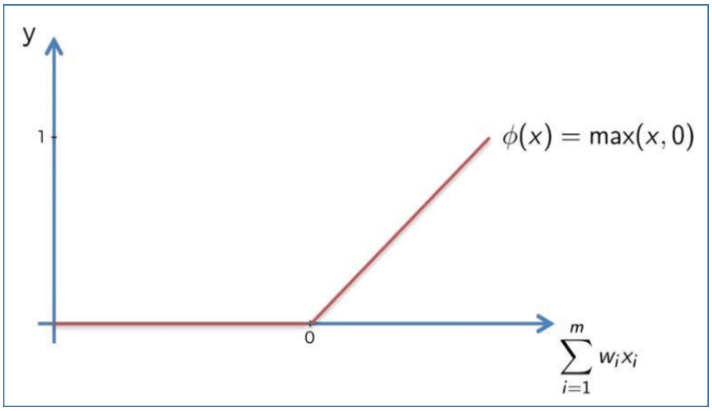

Figure11 The Rectified Linear Unit (ReLU) Activation Function[16]

The rectified linear unit (ReLU) activation function is mathematically described as f(x) = max (0, x), where x represents the input to a neuron. The function returns the input unchanged if it is positive and returns zero otherwise[16]. The ReLU function exhibits linearity for positive inputs and zero output for negative inputs, hence providing non-linearity to the neural network.

One of the primary benefits of the Rectified Linear Unit (ReLU) activation function lies in its notable computing efficiency. The computational complexity of the function is quite low since it just involves a comparison and a maximum operation. This simplicity allows for speedier evaluation in contrast to activation functions that are more intricate in nature.

Furthermore, the rectified linear unit (ReLU) function serves as a solution to the issue of the vanishing gradient problem, which poses a challenge to the successful training of deep neural

networks. The Rectified Linear Unit (ReLU) function is designed to prevent the shrinking of gradients during the backpropagation process for positive inputs by deleting negative values and mapping them to zero. This facilitates the attainment of more stable and efficient gradient updates, hence expediting the convergence of deeper networks.

One additional advantage of the Rectified Linear Unit (ReLU) is its capacity to selectively stimulate neurons in a sparse manner. The rectified linear unit (ReLU) function promotes sparsity within the neural network by selectively stimulating just the appropriate neurons, since it sets negative values to zero. The trait of sparsity may result in more efficient and expressive representations, since it involves a reduced number of neurons in the calculation, hence mitigating the potential for overfitting.

Nevertheless, it is essential to acknowledge that the Rectified Linear Unit (ReLU) may not be appropriate for every situation. One constraint that arises is the issue of the "dying ReLU" phenomenon, whereby neurons have the potential to remain permanently inactive for negative inputs, leading to the presence of non-functional neurons that do not contribute to the learning process. This concern may be addressed by using other forms of the Rectified Linear Unit (ReLU) activation function, such as Leaky ReLU or Parametric ReLU. These versions provide a little non-zero output for negative input values.

To summarize, the Rectified Linear Unit (ReLU) activation function has significant prominence as an extensively used activation function inside the TensorFlow Keras API for the purpose of model training. The popularity of this approach in deep learning may be attributed to its simplicity, computational efficiency, and its capability to effectively tackle the vanishing gradient issue. The incorporation of non-linearity via the Rectified Linear Unit (ReLU) activation function allows neural networks to acquire intricate patterns and enhance their training efficacy, hence playing a significant role in the accomplishments of many artificial intelligence (AI) applications.

2.4.2.3 Sigmoidal Activation Function

The sigmoidal activation function is a frequently used activation function inside the TensorFlow Keras API for the purpose of training models. The function has a smooth, sigmoidal shape and effectively transforms input values into a bounded range of 0 to 1. This property makes it well-suited for binary classification problems and facilitates the generation of probabilistic interpretations.

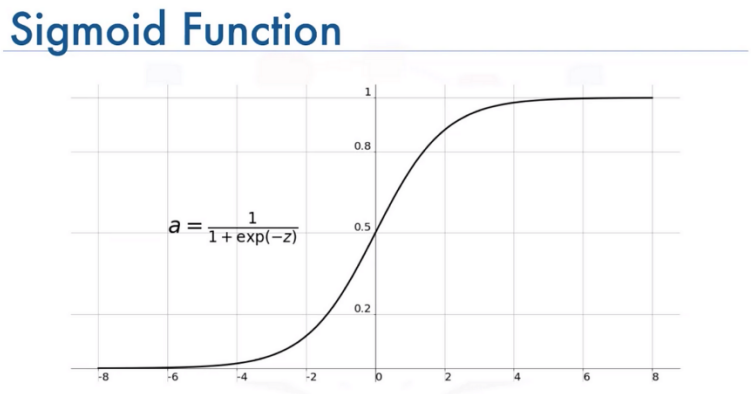

Figure12 The Sigmoidal Activation Function[17]

The sigmoid function may be mathematically expressed as f(x) = 1 / (1 + exp(-x)), where x represents the input to a neuron[18]. The function accepts inputs of real numbers and maps them to a range of values from 0 to 1. As the input tends towards positive infinity, the sigmoid function converges to a value of 1, but as the input tends towards negative infinity, it converges to a value of 0.

The primary benefit of the sigmoid function is in its capacity to compress the input values into a range that resembles probabilities. This characteristic makes it advantageous for binary classification tasks, as the resulting output may be understood as the likelihood of membership to a certain class. In the context of logistic regression modelling, it is customary to use the sigmoid function as a means to generate a probability estimation for the positive class.

In addition, it is important to note that the sigmoid function has differentiability, a key characteristic necessary for the process of backpropagation in the training of models. Efficient computation of gradients enables the implementation of effective weight changes and optimization techniques.

Nevertheless, it is important to acknowledge some constraints associated with the use of the sigmoid function. One of the obstacles that researchers face in the field is sometimes referred to as the "vanishing gradient" issue. In the case of extreme input values, the derivative of the sigmoid function approaches zero, resulting in potential challenges such as sluggish convergence or gradient saturation during the backpropagation process. This problem may hamper the training of deep neural networks.

Additionally, it should be noted that the sigmoid function does not exhibit symmetry with respect to zero, a characteristic that might potentially generate biases inside the neural network. The sigmoid function consistently produces positive output, indicating that the neuron's output is inclined towards positive values.

As a result of these constraints, contemporary deep learning architectures generally prioritize alternate activation functions such as Rectified Linear Unit (ReLU) and its variations. Nevertheless, the sigmoid function continues to be used in some contexts, particularly in the concluding layer of a binary classification model when there is a need for a probabilistic understanding of the output.

To summarize, the sigmoidal activation function used in the TensorFlow Keras API is a continuous and differentiable function that exhibits an S-shaped curve. Its primary purpose is to transform input values into a bounded range spanning from 0 to 1[18]. The method is often used in jobs involving binary classification and offers probabilistic interpretations. Despite its shortcomings, such as the vanishing gradient issue and bias towards positive values, the sigmoid function continues to have relevance in certain circumstances that need a sigmoidal output and probabilistic interpretations.

2.4.2.4 Optimizer: Adam algorithm

The optimizer plays a key role in the TensorFlow Keras API for model training since it governs the manner in which the neural network's weights are adjusted during the training procedure. The aforementioned factor plays a crucial part in the reduction of the loss function and enhancement of the model's overall performance. The optimizer utilizes a range of optimization methods to repeatedly modify the weights by using the gradients calculated during the backpropagation process.

The Adam (Adaptive Moment Estimation) technique is a commonly used optimizer in TensorFlow Keras. Adam is a novel optimization algorithm that integrates the advantageous features of two existing algorithms, AdaGrad and RMSprop, in order to provide effective and adaptable weight updates.

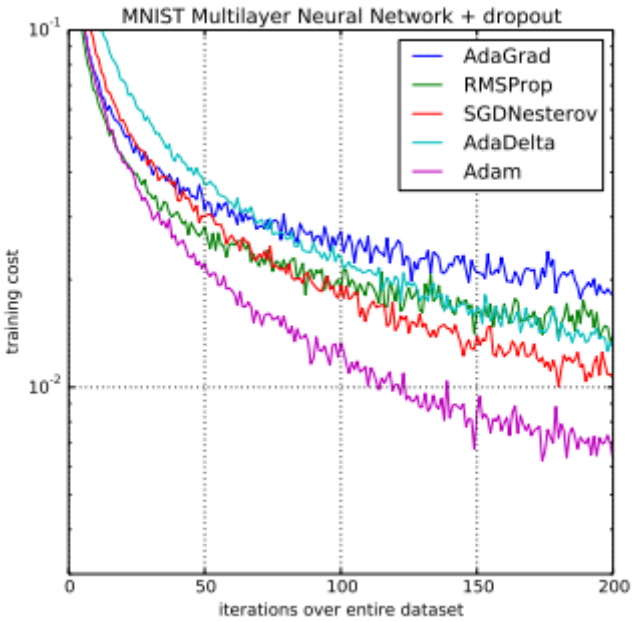

Figure13 The Comparison of Optimization Algorithm[17]

The Adam method is designed to optimize the learning process by adjusting the learning rate for each weight parameter. This adjustment is made based on the historical gradients of the weight parameter. The algorithm computes the adaptive learning rates by using a blend of the first and second moments of the gradients. The first moment corresponds to the mean value

of the gradients, whereas the subsequent moment corresponds to the mean value of the squared gradients.

The use of an adjustable learning rate enables the algorithm to effectively handle diverse parameter types. This approach facilitates accelerated convergence by implementing more substantial changes to parameters associated with lower gradients, while applying smaller updates to parameters associated with greater gradients. The adaptive nature of this behaviour enables the mitigation of the constraints imposed by fixed learning rates, making Adam a very suitable choice for a diverse array of optimization issues.

Furthermore, Adam employs a bias correction technique to mitigate the bias towards zero that may arise during the first phases of training. This adjustment addresses the issue of bias towards zero in the estimations of the first and second moments during the first stages of training. By taking into consideration this bias, Adam is able to obtain updates that are more accurate and enhance the process of convergence.

The Adam algorithm has gained significant popularity in the field of deep learning as a result of its strong performance and user-friendly nature. The learning rate is adjusted automatically with minimum need for human adjustment of hyperparameters. This technology is very compatible with many neural network topologies and training settings.

The integration of the Adam optimizer into the model training process is facilitated by using the TensorFlow Keras API. When the optimizer 'adam' is specified during the compilation of the model, the algorithm is used to perform weight updates throughout the training process. The Adam optimizer offers many hyperparameters, including the learning rate, decay rates, and epsilon, which may be fine-tuned to meet the unique demands of the given job.

In brief, the optimizer inside the TensorFlow Keras API is tasked with the responsibility of modifying the weights of a neural network during the process of training. The Adam method is widely used as an optimizer in many machine learning tasks. It incorporates adaptive

learning rates, first and second moment estimates, and bias correction techniques to rapidly and effectively update the weights[18]. Due to its adaptable nature and strong performance, this approach is extensively used in the field of deep learning to optimize neural network models.

2.4.2.5 Loss Function: Binary Crossentropy

Within the TensorFlow Keras API for training models, the loss function plays a vital role in quantifying the disparity between the anticipated output of a neural network and the actual target values. The metric functions as an indicator of the model's performance and provides guidance for the optimization process during the training phase.

The loss function computes a singular scalar quantity that quantifies the disparity between the expected output and the actual ground truth. The objective of the optimization procedure is to reduce the aforementioned loss value, thereby enhancing the performance of the model.

Binary Crossentropy is a frequently used loss function in TensorFlow Keras for jobs involving binary classification. This particular design is intended for scenarios in which the desired outcome is binary, such as the classification of a picture as either a cat or a dog.

The concept of Binary Crossentropy is used to quantify the discrepancy between the expected probability distribution and the actual probability distribution. The function computes the mean cross-entropy loss across all instances in the dataset. The Binary Crossentropy loss function may be mathematically stated as follows[19]:

$$L(y\_true, y\_pred) = -(y\_true * \log(y\_pred) + (1 - y\_true) * \log(1 - y\_pred))$$

The variable y_true denotes the actual goal values, which may take the values of either 0 or 1. On the other hand, y_pred represents the anticipated probability.

The Binary Crossentropy loss function imposes a penalty on significant disparities between the expected probability and the actual labels. When the projected likelihood is in close proximity to the real label, the resulting loss value will be minimal. On the other hand, when the anticipated likelihood diverges much from the actual label, the loss value will be substantial.

During the training phase, the optimizer utilizes the loss value to modify the weights and biases of the model, with the objective of minimizing the loss function and enhancing the accuracy of the model.

It is noteworthy to mention that the Binary Crossentropy loss function is only one of many alternative choices inside the TensorFlow Keras framework[20]. The appropriateness of various loss functions may vary depending on the specific situation being addressed. In the context of multi-class classification problems, it is common practice to use Categorical Crossentropy as the preferred loss function.

The Binary Crossentropy loss function may be used in the TensorFlow Keras API by specifying it as the loss parameter during the compilation of the model. The loss function plays a crucial role in the training of neural networks as it serves as a guiding factor for the optimization process and facilitates the acquisition of accurate prediction capabilities by the model.

2.4.3 Implementation

The objective of training the MLPF model with this particular configuration and using a varied training dataset is to enhance the model's ability to accurately discern events as either signal or noise. The model acquires the ability to use the polarity and time difference information obtained from the patch of neighbouring data points in order to effectively make precise filtering choices.

The research methodology incorporated a Multi-Layer Perceptron Feedforward (MLPF) configuration with a patch size of 7x7. Age values were scaled using sigmoidal activation, and a correlation time of 10000 was employed. The MLPF model consists of a hidden layer with 20,000 neurons, utilizing Rectified Linear Unit (ReLU) activation. Sigmoidal activation is applied in the output layer. The optimization was performed using the Adam optimizer, and the loss function employed is Binary Crossentropy. The training dataset comprised 100,000 event samples that include a combination of noises and signal-noise that are already identified.

2.5 The Conversion of Event Data into Video

The process of transforming event data captured by an event camera into a video output necessitates the use of computational techniques to analyse the continuous stream of asynchronous events and reconstruct a series of frames that may be presented in a video format. The event data obtained from an event camera comprises discrete events, with each event including information such as the coordinates of the event, the polarity of the event, and the timestamp at which the event occurred. In order to transform this data into a visual output, a series of procedures are normally undertaken.

For the final coding for converting event data to video, please take reference from the Appendix VII.

3. Results and Discussion

The True Positive Rate (TPR), sometimes referred to as sensitivity or recall, is the ratio of properly identified positive signals by the BAF to the total number of genuine positive signals. A higher true positive rate (TPR) signifies a greater number of signals being accurately detected and preserved subsequent to the application of the filter.

The False Positive Rate (FPR) refers to the ratio of negative sounds that are erroneously identified as signals by the BAF. Put simply, it quantifies the frequency at which false alarms or extraneous sounds are erroneously classified as signals. A lower false positive rate (FPR) signifies that the filter allows fewer instances of noise to get through as false positives.

3.1 The Result of Background Activity Filter (BAF)

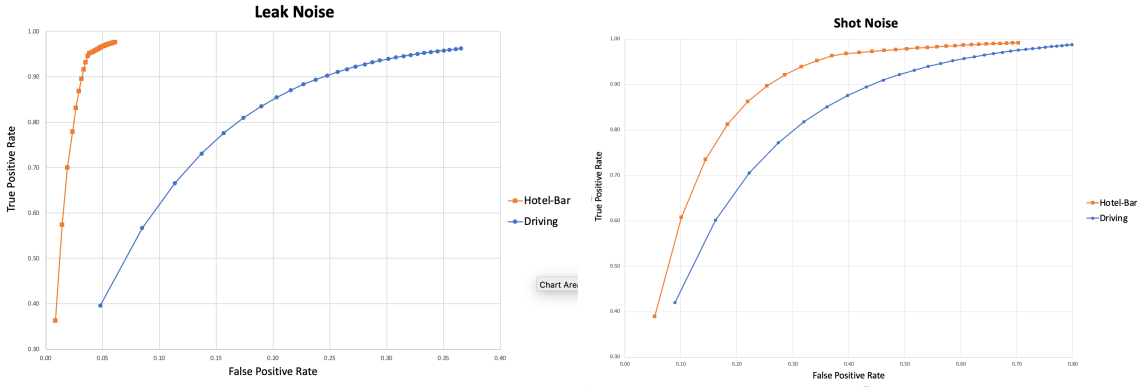

Figure14 The ROC Curve Result of BAF for Different Datasets

Based on the ROC curve result shown in Figure13, it has been noted that the BAF exhibits superior performance in effectively filtering leak signals and minimizing the occurrence of false positives for the hotel-bar dataset since with TPR(around 0.9), the datasets with leak noises always have smaller FPR comparing with the datasets with shot noises. This implies that the BAF demonstrates enhanced efficacy in the detection and preservation of authentic signals, while concurrently mitigating the occurrence of false positives or extraneous disturbances within the filtered output.

Furthermore, it has been noted that the BAF is better suited for stationary cameras. This conclusion may be deduced from the observation that the BAF exhibited a higher TPR and a lower FPR in the context of the stationary camera dataset, namely the hotel-bar scenario. The BAF's capacity to effectively preserve genuine signals while minimizing the occurrence of false positives makes it a very advantageous option in situations when the camera's perspective stays generally stable.

It is vital to acknowledge that the appropriateness and efficacy of a filter may fluctuate based on the distinct attributes of the dataset and the filtering criteria. The examination of TPR and FPR aids in comprehending the merits and limitations of the BAF in various situations, facilitating well-informed decision-making when choosing the suitable filter for a certain application.

3.2 The Result of Spatiotemporal Correlation Filter (STCP)

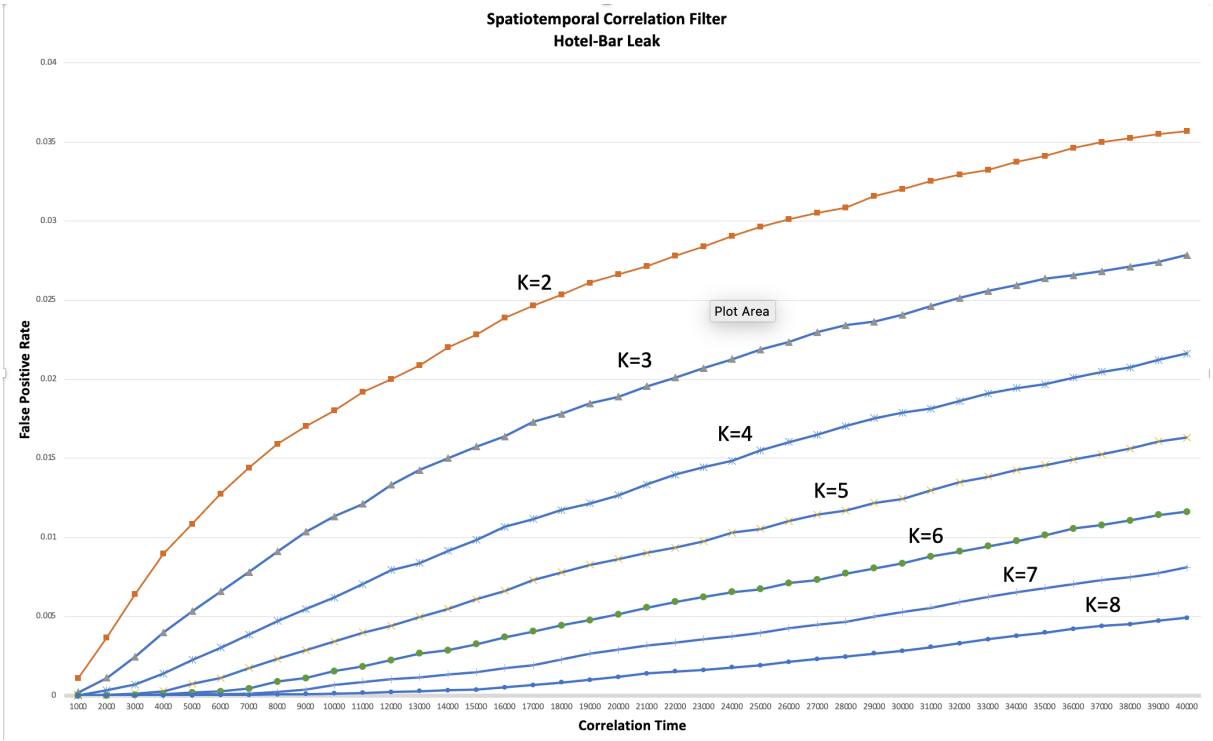

Figure15 The Result of STCP for Hotel-bar Dataset with Leak Noises

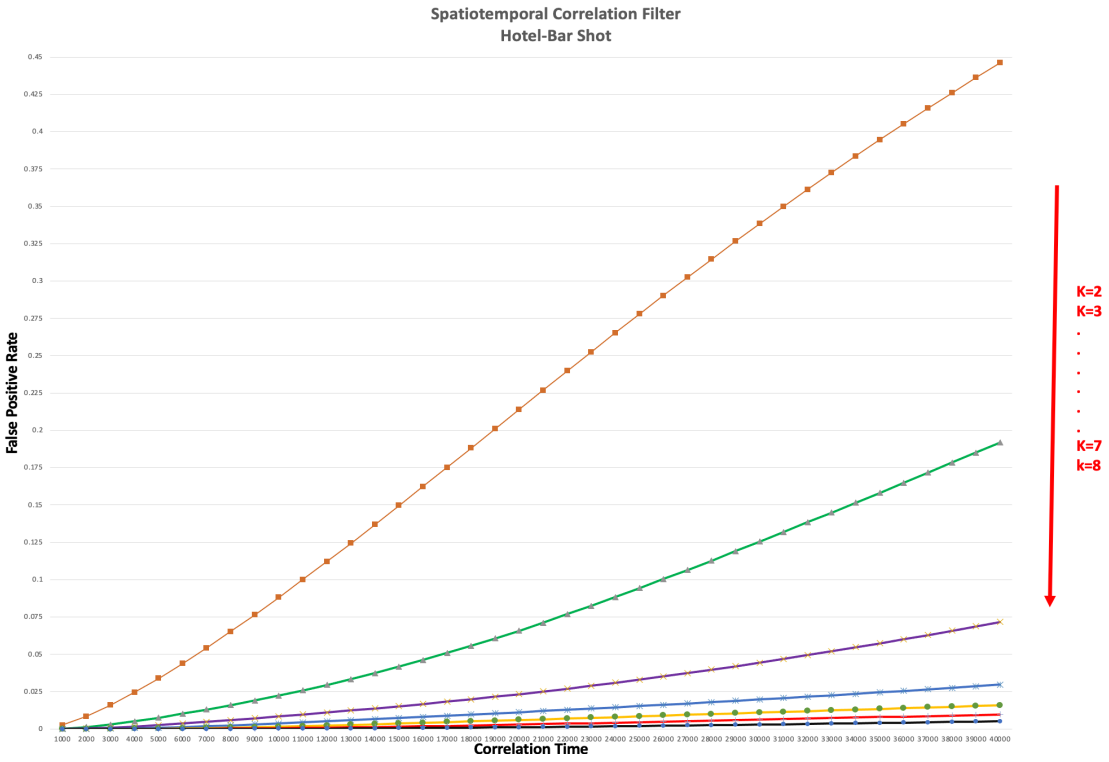

Figure16 The Result of STCP for Hotel-bar Dataset with Shot Noises

Based on the findings obtained from the STCF analysis conducted on the datasets pertaining to the hotel bar leak test, the following observations have been made:

The impact of correlation time τ is seen when an increase in τ, while maintaining a constant value of K (The number of neighbours), leads to a greater passage of noise through the filter. This suggests that as the correlation time increases, there is a greater probability of preserving noise in the filtered output. This implies that, within the unique context of the hotel bar leak test datasets, a longer correlation time may not be as efficient in differentiating between signals and sounds.

The influence of the number of neighbours, denoted as K, on the filtering efficiency may be seen. When the correlation time is constant, an increase in K leads to a decrease in the false positive rate. This suggests that a higher value of K improves the effectiveness of the filtering process. This implies that including a greater number of neighbouring instances and their temporal disparities has the potential to enhance the precision in distinguishing occurrences

as either signals or noises. When the value of K is raised while keeping the correlation time constant, the STCF has superior performance in terms of eliminating false positives.

Moreover, it has been noted that comparable outcomes have been obtained for the datasets pertaining to hotel bar shots. This suggests that the observed performance patterns are constant across various dataset categories.

The results indicate that, within the dataset of hotel bars, there is no clear evidence to support the notion that raising the correlation time would lead to enhanced filtering performance of the STCF. Nevertheless, the augmentation of the number of neighbours K might potentially enhance performance by mitigating the occurrence of false positives.

It is noteworthy that the aforementioned findings pertain only to the datasets of hotel bars and the distinctive attributes of the events undergoing filtration. The efficacy of the STCF technique may exhibit variability across diverse contexts and datasets. Hence, it is important to assess the efficacy of the filtering process by considering certain criteria and dataset attributes in order to ascertain the most suitable setup for the STCF or any other filtering algorithm.

3.3 The Result of Multilayer Perceptron Denoising Filter (MLPF)

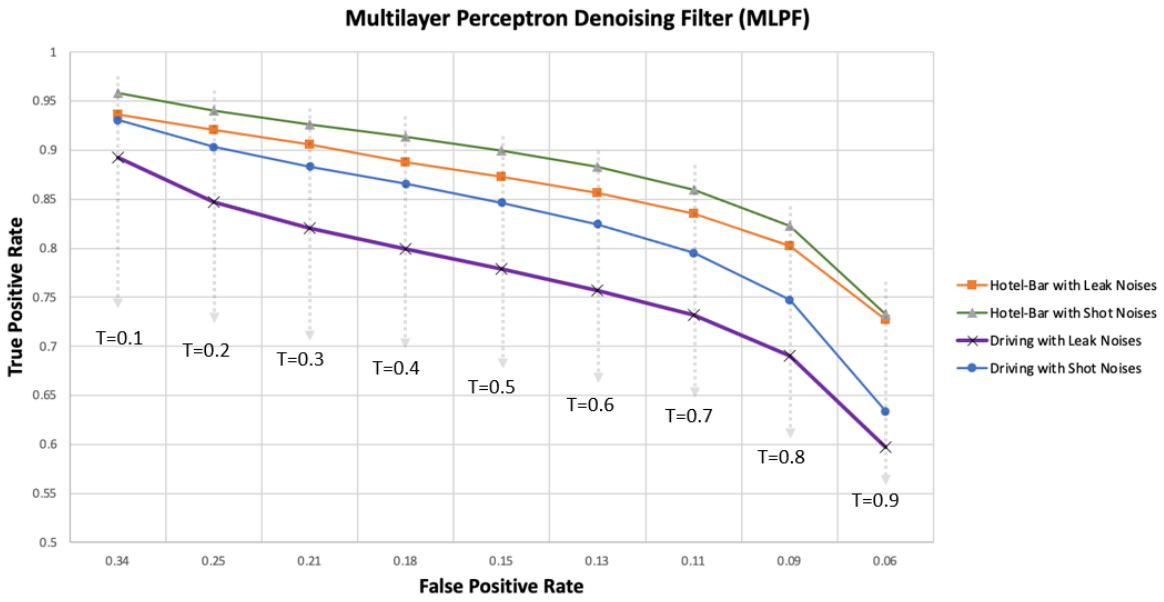

Figure17 The ROC Curve Result of BAF for Different Datasets

Based on the findings obtained from the MLPF analysis, it can be noticed that the MLPF demonstrates effective performance in filtering both leak sounds and shot noises. Furthermore, a comparative analysis was conducted to evaluate the filtering efficacy across various thresholds, revealing that a lower threshold yields superior filtering outcomes.

The threshold inside the MLPF algorithm is responsible for determining the degree of statistical significance for the temporal disparity between the present event and its neighboring events. By reducing the threshold value, the Multi-Layer Perceptron Filter (MLPF) adopts a more rigorous approach in distinguishing events as either signal or noise, leading to enhanced performance in the filtering process.

It is noteworthy that the Multi-Layer Perceptron Filter (MLPF) exhibits commendable performance in the study conducted, effectively mitigating both leak and shot sounds. This implies that the Multilayer Perceptron Neural Network (MLPF) demonstrates efficacy in collecting the patterns and properties of both forms of noise, hence facilitating precise filtering.

The adaptability and promise for strong noise filtering of the MLPF are underscored by its ability to function well across many forms of noise. Nevertheless, it is crucial to acknowledge that the ideal threshold value might potentially differ based on the distinct dataset and features of the noise. Optimizing the threshold parameter in accordance with the particular needs and dataset is crucial in order to get optimal filtering performance.

In conclusion, the Multi-Layer Perceptron Filter (MLPF) demonstrates favorable performance in effectively attenuating both leak sounds and shot noises. Research has shown that using smaller threshold values leads to improved filtering outcomes, underscoring the need of carefully adjusting the threshold parameter to get optimum performance.

3.4 The Comparison of Three Filters' Performance

Hotel-Bar Leak	k=2		k=3		k=4	
Correlation Time(us)	FPR	TPR	FPR	TPR	FPR	TPR
38000	0.0353	0.9510	0.0271	0.9085	0.0207	0.8332
39000	0.0355	0.9519	0.0274	0.9103	0.0212	0.8366
40000	0.0357	0.9528	0.0278	0.9121	0.0216	0.8398

Driving Leak	k=2		k=3		k=4	
Correlation Time(us)	FPR	TPR	FPR	TPR	FPR	TPR
38000	0.2925	0.9165	0.2287	0.8523	0.1711	0.7614
39000	0.2968	0.9184	0.2325	0.8553	0.1750	0.7655
40000	0.3005	0.9201	0.2362	0.8581	0.1792	0.7695

Hotel-Bar Shot	k=2		k=3		k=4	
Correlation Time(us)	FPR	TPR	FPR	TPR	FPR	TPR
38000	0.4261	0.9790	0.1784	0.9480	0.0658	0.8919
39000	0.4363	0.9798	0.1851	0.9498	0.0687	0.8952
40000	0.4462	0.9806	0.1918	0.9515	0.0717	0.8983

Driving Shot	k=2		k=3		k=4	
Correlation Time(us)	FPR	TPR	FPR	TPR	FPR	TPR
38000	0.6033	0.9655	0.3882	0.9148	0.2501	0.8370
39000	0.6128	0.9671	0.3970	0.9178	0.2567	0.8415
40000	0.6219	0.9685	0.4056	0.9206	0.2634	0.8459

Fair Performance

Still suitable For Stationary Camera application

Better Performance for filtering Shot Noises (compare with BAF(t=3000) [FPR:0.34,TPR:0.95])

Figure18 The Result of STCP for Comparison

Based on the data presented, it can be inferred that the STCF is also applicable for stationary camera applications. This suggests that the STCF has favorable efficacy in event filtration and signal-noise discrimination, even in situations when the camera stays motionless.

When comparing the performance of two shot noise filtering techniques, BAF and STCF, with same true positive rate (around 0.95), it was discovered that STCF exhibited a reduced FPR (0.19) in comparison to BAF (0.34). This finding suggests that the STCF method demonstrates superior performance compared to the BAF method in effectively filtering out shot sounds. It does this by striking a more optimal equilibrium between accurately detecting genuine signals and decreasing the occurrence of false alarms or extraneous noises.

The results demonstrate that the use of STCF is efficacious in stationary camera scenarios and offers superior performance in mitigating shot sounds compared to BAF, as seen by the decreased incidence of false positives.

Based on the conducted investigation, a comparison has been made between the performance of three filtering algorithms, namely STCF, BAF, and MLPF, in the context of filtering both leak sounds and shot noises. The following are the conclusions that have been derived:

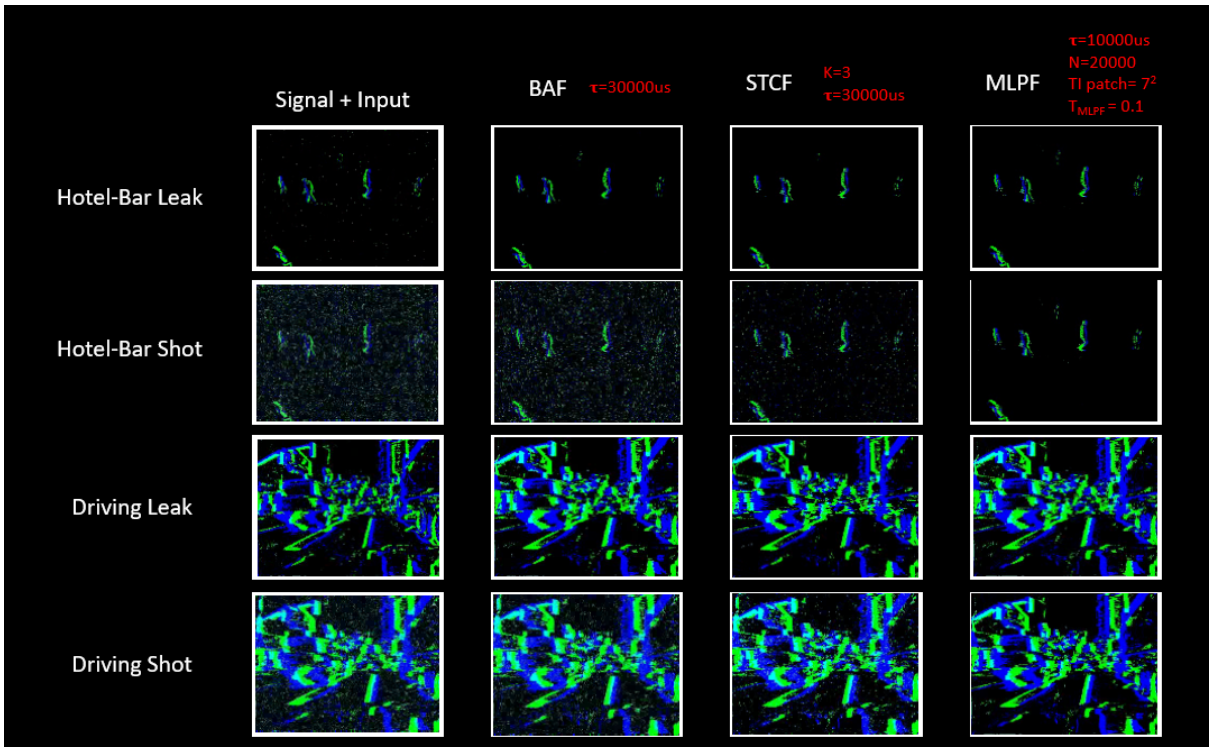

Figure19 The Result of BAF, STCP and MLPF for Comparison

The use of three algorithms, namely STCF, BAF, and MLPF, is viable for the purpose of filtering leak sounds. It is evident that all three algorithms exhibit satisfactory performance in this aspect, but with potential variations in their individual performance characteristics. Conducting a more extensive examination and comparison of the algorithms' performance, specifically in relation to metrics such as true positive rate, false positive rate, and other pertinent measurements, would provide a full comprehension of their respective merits and limitations.

In the context of filtering shot noises, it has been determined that the STCF algorithm exhibits superior performance in comparison to the BAF method. This implies that the STCF algorithm exhibits more efficacy in discriminating shot sounds from signals, hence yielding a reduced incidence of false positives.

According to the result, it can be concluded that the MLPF algorithm exhibits the most superior performance when compared to the other two filtering algorithms in terms of overall effectiveness. The Multi-Layer Perceptron Filter (MLPF) has superior performance in terms of filtering efficacy when compared to both the Short-Time Fourier Transform Filter (STCF) and the Bandpass Adaptive Filter (BAF), taking into account the mitigation of both leak sounds and shot noises.

5. Conclusion

It is important to acknowledge that the inferences derived from the analysis are contingent upon the particular comparisons conducted and the dataset or parameters used. The efficacy of filtering algorithms may fluctuate based on the distinct attributes of the dataset, variations in noise kinds, and the unique demands of the application. Hence, it is important to conduct a full evaluation of filtering algorithms and take into account many performance measures in order to ascertain the best appropriate algorithm for a given circumstance. The use of suitable visualizations or performance measures might facilitate the comprehension and comparison of individuals' efficacy in various noise filtering jobs.

And without doubt, the design of this research still have drawbacks. For improvement, to enhance the comparative analysis of BAF and STCF, it is advantageous to augment the size and diversity of the datasets used for testing purposes. This approach would provide a more thorough assessment of the algorithms and their efficacy in various contexts.

To improve the accuracy of the MLPF model, it is advisable to acquire a larger volume of training data. Enhancing the scale and heterogeneity of the training dataset has the potential to enhance the model's capacity to generalize and effectively discern extraneous signals.

The investigation of adjustable factors is being undertaken, whereby specific tunable variables such as patch size, correlation duration, and the amount of neurons in the classifier have been predetermined in your particular scenario. Conducting an investigation into the effects of manipulating these factors on the effectiveness of the filtering process might provide intriguing results. Through the process of conducting experiments with various values for these factors, one may evaluate the potential impact of adjusting them on the overall quality of filtering outcomes.

Further Investigation of the MLPF Algorithm: Considering the favorable outcomes seen in the examination of the MLPF algorithm, it is advisable to conduct a more extensive exploration of its capabilities. This may include performing comprehensive tests and assessments, refining the adjustable variables, and tweaking the MLPF model to enhance the efficacy of the filtering process.

By considering these aspects, one may improve the resilience and efficiency of the filtering algorithms and get a more comprehensive comprehension of their capabilities. The ongoing enhancement and investigation of these algorithms will play a significant role in the advancement of noise filtering methods in the future.

Appendices

Appendix I The Python Code for Reading Event Test Data from TXT File

```
class Events(object):
    def __init__(self, num_events: int, width: int, height: int) -> np.ndarray:
        # events contains the following index:
        # t: the timestamp of the event.
        # x: the x position of the event.
        # y: the y position of the event.
        # p: the polarity of the event.
        self.events = np.zeros((num_events), dtype=[("t", np.uint64), ("x", np.uint16), ("y", np.uint16), ("p", np.bool_), ("s", np.bool_)])
        self.width = width
        self.height = height
        self.num_events = num_events

# Read event data from file
def process_text_file(filename: str) -> Events:
    with open(filename, 'r',buffering=4000000) as f:
        num_events = 0
        for _ in f:
            num_events += 1


    print("!!!!!",num_events)
    events = Events(num_events, WIDTH, HEIGHT)

    with open(filename, 'r',buffering=4000000) as f:
        for i, line in enumerate(tqdm(f)):
            event = line.split('\t')
            if len(event) != 5: print("!!!!!!!!!")
            assert len(event) == 5
            events.events[i]["x"], events.events[i]["y"],events.events[i]["t"], =  int(event[0])-1, int(event[1])-1,int(event[3])   #*1e6
            events.events[i]["p"] = True if int(event[2]) == 1 else False
            events.events[i]["s"] = int(event[4])
            #if(i==100): break

    return events
```

Appendix II The Python Code for Background Activity Filtering

```
# Nearest Neighbourhood/Background Activity Filtering
def background_activity_filter(event_array: Events, time_window: int=200):
    TP,TN,FP,FN=0,0,0,0
    max_x, max_y = event_array.width - 1, event_array.height - 1
    t0 = np.ones((event_array.height, event_array.width)) - time_window - 1
    x_prev, y_prev, p_prev = 0, 0, 0
    valid_indices = np.ones(event_array.num_events, dtype=np.bool_)

    for i, e in tqdm(enumerate(event_array.events)): #tqdm: process bar;    // i is the index, e are thhe content(will go through each)
        ts, x, y, p = e["t"], e["x"], e["y"], e["p"]

        if x_prev != x or y_prev != y or p_prev != p: #if install the first event, then not go into if () condition
            t0[y][x] = -time_window

            min_x_sub = max(0, x-1)
            max_x_sub = min(max_x, x+1)
            min_y_sub = max(0, y-1)
            max_y_sub = min(max_y, y+1)

            t0_temp = t0[min_y_sub:(max_y_sub+1), min_x_sub:(max_x_sub + 1)]
            if min(ts - t0_temp.reshape(-1, 1)) > time_window:
                valid_indices[i] = 0 #indixcate each event to tell nosie or signal
            if valid_indices[i]==1 and e["s"]==1: TP+=1
            if valid_indices[i]==0 and e["s"]==0: TN+=1
            if valid_indices[i]==1 and e["s"]==0: FP+=1
            if valid_indices[i]==0 and e["s"]==1: FN+=1
        t0[y][x], x_prev, y_prev, p_prev = ts, x, y, p #always update the timestamp
    print()
    print("TP",TP)
    print("TN",TN)
    print("FP",FP)
    print("FN",FN)
    return event_array.events[valid_indices], np.count_nonzero(valid_indices == True)
```

Appendix III The Python Code for Spatiotemporal Correlation Filtering

```
#  Spatiotemporal Correlation Filtering (STCF)
def Spatiotemporal_Correlation_Filter(event_array: Events, time_window: int=200, k: int=1):
    TP,TN,FP,FN=0,0,0,0
    max_x, max_y = event_array.width - 1, event_array.height - 1
    t0 = np.ones((event_array.height, event_array.width)) - time_window - 1
    x_prev, y_prev, p_prev= 0, 0, 0,
    valid_indices = np.ones(event_array.num_events, dtype=np.bool_)

    for i, e in tqdm(enumerate(event_array.events)): #tqdm: process bar;    // i is the index, e are thhe content(will go through each)
        count=0
        ts, x, y, p = e["t"], e["x"], e["y"], e["p"]

        if x_prev != x or y_prev != y or p_prev != p: #if install the first event, then not go into if () condition
            t0[y][x] = -time_window
            min_x_sub = max(0, x-1)
            max_x_sub = min(max_x, x+1)
            min_y_sub = max(0, y-1)
            max_y_sub = min(max_y, y+1)

            t0_temp = t0[min_y_sub:(max_y_sub+1), min_x_sub:(max_x_sub + 1)]
            for c in (ts-t0_temp.reshape(-1,1)):
                if c<= time_window: count+=1


            if count< k:
                valid_indices[i] = 0 #indixcate each event to tell nosie or signal
            if valid_indices[i]==1 and e["s"]==1: TP+=1
            if valid_indices[i]==0 and e["s"]==0: TN+=1
            if valid_indices[i]==1 and e["s"]==0: FP+=1
            if valid_indices[i]==0 and e["s"]==1: FN+=1

        t0[y][x], x_prev, y_prev, p_prev = ts, x, y, p #always update the timestamp


    with open('/content/drive/MyDrive/myfile.txt', 'a') as f:
        s=str(TP)+'\t'+str(TN)+'\t'+str(FP)+'\t'+str(FN)+'\n'
        f.writelines(s)

    return event_array.events[valid_indices], np.count_nonzero(valid_indices == True)
```

Appendix IV The Python Code for Getting Event Data from Testing Datasets

```
def Get_Training_Data(event_array: Events, time_window: int=200,S_MLPF:int=49):
    max_x, max_y = event_array.width - 1, event_array.height - 1
    patch = np.zeros((S_MLPF),dtype=[("p", np.uint16), ("a", np.float16)])
    TI = np.ones((event_array.height, event_array.width),dtype=[("p", np.uint16), ("t", np.uint32)])
    x_prev, y_prev, p_prev= 0, 0, 0,
    valid_indices = np.ones(event_array.num_events, dtype=np.bool_)
    pat_c,label_c=-1,-1
    pat,label,aa=[],[],[]


    for count, e in tqdm(enumerate(event_array.events)): #tqdm: process bar;    // i is the index, e are thhe content(will go through each)
        ts, x, y, p, s= e["t"], e["x"], e["y"], e["p"], e["s"]
        if x_prev != x or y_prev != y or p_prev != p: #if install the first event, then not go into if () condition
            aa=int((math.sqrt(S_MLPF)-1)/2)
            min_x_sub = max(0, x-aa)
            max_x_sub = min(max_x, x+aa)
            min_y_sub = max(0, y-aa)
            max_y_sub = min(max_y, y+aa)
            #if count>19700 and count<20010: print(count,ts)
            TI_temp = TI[min_y_sub:(max_y_sub+1), min_x_sub:(max_x_sub + 1)]
            #update PATCH
            #if count>19700 and count<20010: print(TI_temp)
            for i,c in (enumerate(TI_temp.reshape(-1,1))):
              if (ts-c["t"])>(time_window) :
                patch[i]["a"]=0
                patch[i]["p"]=0
              else:
                patch[i]["p"]=1 if c["p"]==p else 2
                patch[i]["a"]=1-float(float(ts-c["t"])/time_window)
              TI[y][x]["t"] = e["t"]
              TI[y][x]["p"] = e["p"]
            #if count>19700 and count<20010:print(patch)
        pat.append(patch.tolist())
        del patch
        patch = np.zeros((S_MLPF),dtype=[("p", np.uint16), ("a", np.float16)])

        if s==True:
          label.append([1])
        else:
          label.append([0])
        #if count>3000000: break;
    return pat,label
```

Appendix V The Python Code for Building Classifier

```
hidden=20000
epochs=10
model=Sequential()
model.add(Dense(units=hidden,  activation='relu',input_dim=(len(x_train.columns))))
model.add(Dense(units=1, activation='sigmoid'))
model.summary()
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train,y_train,epochs=epochs,batch_size=30)
model.save('/content/drive/MyDrive/training data/7x7_model/')
```

Appendix VI The Python Code for Loading the Classifier and Getting Result for Multilayer Perceptron Filtering

```
loaded_model = tf.keras.models.load_model('/content/drive/MyDrive/training data/7x7_model/')
loaded_model.summary()
Y_PRE=loaded_model.predict(x_train)
y_pre=Y_PRE
y_pre=[0 if val<0.1 else 1 for val in y_pre]
print(accuracy_score(y_train,y_pre))
tn, fp, fn, tp = confusion_matrix(y_train, y_pre).ravel()
print(confusion_matrix(y_train, y_pre).ravel())
TPR = tp * 1.0 / (tp + fn) #the number of signal leave
print('TPR:',TPR)
FPR = fp * 1.0 / (tn + fp) #the number of noice leave
print('FPR:',FPR)
```

Appendix VII Convert Event Data into Video

```python
# Slice all events based on time window
def time_window_slice(event_array: Events, time_window: int=33000, drop_incomplete: bool=True) -> list:
    timestamps = event_array.events["t"]

    if drop_incomplete:
        num_slices = int(np.floor(((timestamps[-1] - timestamps[0]) - time_window)/time_window) + 1)
    else:
        num_slices = int(np.ceil(((timestamps[-1] - timestamps[0]) - time_window)/time_window) + 1)

    window_start_time = np.arange(num_slices) * time_window  + timestamps[0]
    window_end_time = window_start_time + time_window
    indices_start = np.searchsorted(timestamps, window_start_time) # the index of the first element that less than window_start_time
    indices_end = np.searchsorted(timestamps, window_end_time) # the index of the first element that window_end_time

    return [event_array.events[indices_start[i]:indices_end[i]] for i in range(num_slices)]

# Generate frames based on sliced events
def accumulate_and_generate(sliced_event_array: list, width, height) -> np.ndarray:
    frames = np.zeros((len(sliced_event_array), height, width, 2), dtype=np.uint8)
    #mask = np.zeros((height, width, 2))

    for i, e in tqdm(enumerate(sliced_event_array)): #: number of frame
        for _, x, y, p in e:
            frames[i][y][x][int(p)] = 255

    return frames

def addChannel(image):
    no_of_frames = image.shape[0]
    dummy_channel = np.zeros((no_of_frames, HEIGHT, WIDTH, 1), dtype=np.uint8)
    print(dummy_channel.shape)
    image = np.append(image, dummy_channel, axis=3)
    print(image.shape)

    return image

def save_video(frames, mode: str) -> None:
    # Store the output as a video
    img_array = []
    height, width, _ = frames[0].shape
    size = (width, height)
    for f in frames:
        img_array.append(f)

    vid_recorder = cv2.VideoWriter(OUTPUT + mode + '.avi', cv2.VideoWriter_fourcc(*'DIVX'), 15, size)

    # For certain dataset like DDD20, where the address definition is flipped 180deg
    # for i in img_array:
    #     i = np.rot90(i, 2)
    #     vid_recorder.write(i)

    # Otherwise
    for i in img_array:
        vid_recorder.write(i)

    vid_recorder.release()
```

References:

[1]iniVation, “DAVIS 346,” https://inivation.com, Aug. 15, 2019. https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf.

[2]Scheerlinck, Cedric & Rebecq, Henri & Stoffregen, Timo & Barnes, Nick & Mahony, Robert & Scaramuzza, Davide. (2019). CED: Color Event Camera Dataset.

[3]What Are Event-Cameras - Event-Based Vision And Event Camera SLAM,” Dioram: Computer Vision, Machine Learning, SLAM for AR/VR, robots, drones and autonomous vehicles, Oct. 04, 2022. https://dioramslam.com/2021/10/08/event-cameras/

[4]OPTICA, “Event Cameras: A New Imaging Paradigm,” OPTICA, Jul. 2022. Accessed: Nov. 24, 2023. [Online]. Available: https://www.optica-opn.org/opn/media/Images/PDF/2022/07_0822/048-055_OPN_0708_22.pdf?ext=.pdf

[5]Czech, Daniel & Orchard, Garrick. (2016). Evaluating noise filtering for event-based asynchronous change detection image sensors. 19-24. 10.1109/BIOROB.2016.7523452.

[6]V. R. Padala, A. Basu, and G. Orchard, “A Noise Filtering Algorithm for Event-Based Asynchronous Change Detection Image Sensors on TrueNorth and Its Implementation on TrueNorth,” Frontiers in Neuroscience, Mar. 05, 2018. https://doi.org/10.3389/fnins.2018.00118

[7]J. Wu, C. Ma, X. Yu and G. Shi, "Denoising of Event-Based Sensors with Spatial-Temporal Correlation," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 4437-4441, doi: 10.1109/ICASSP40776.2020.9053002

[8]Sensors Group, Inst. of Neuroinformatics, “Unraveling the paradox of intensity-dependent DVS pixel noise,” Sep. 17, 2021. https://arxiv.org/pdf/2109.08640.pdf .

[9]C. Li, L. Longinotti, F. Corradi and T. Delbruck, "A 132 by 104 10µm-Pixel 250µW 1kefps Dynamic Vision Sensor with Pixel-Parallel Noise and Spatial Redundancy Suppression," 2019 Symposium on VLSI Circuits, Kyoto, Japan, 2019, pp. C216-C217, doi: 10.23919/VLSIC.2019.8778050.

[10]C. Chan, “What is a ROC Curve - How to Interpret ROC Curves - Displayr,” Displayr, Aug. 23, 2022. https://www.displayr.com/what-is-a-roc-curve-how-to-interpret-it/

[11]S. Guo and T. Delbruck, "Low Cost and Latency Event Camera Background Activity Denoising," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 1, pp. 785-795, 1 Jan. 2023, doi: 10.1109/TPAMI.2022.3152999.

[12]Y. Chen, Y. Huang, F. Li, X. Zeng, W. Li and M. Wang, "Denoising Method for Dynamic Vision Sensor Based on Two-Dimensional Event Density," 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 2023, pp. 1-4, doi: 10.1109/ISCAS46773.2023.10181865.

[13]Emeksiz, Deniz & Temizel, Alptekin. (2012). A continuous object tracking system with stationary and moving camera modes. Proceedings of SPIE - The International Society for Optical Engineering. 8541. 15-. 10.1117/12.973720.

[14]“What’s coming in TensorFlow 2.0.” https://blog.tensorflow.org/2019/01/whats-coming-in-tensorflow-2-0.html

[15]“Keras: The high-level API for TensorFlow,” TensorFlow. https://www.tensorflow.org/guide/keras

[16]Bhurtel, Manish & Shrestha, Jabeen & Lama, Niten & Bhattarai, Sajan & Uprety, Aashma & Guragai, Manoj. (2019). DEEP LEARNING BASED SEED QUALITY TESTER.

[17]J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” MachineLearningMastery.com, Jan. 12, 2021. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[18]Chen, Jinghui & Zhou, Dongruo & Tang, Yiqi & Yang, Ziyan & Cao, Yuan & Gu, Quanquan. (2020). Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks. 3239-3247. 10.24963/ijcai.2020/448.

[19]S. Goyal, “ML Loss Functions: An Overview - Geek Culture - Medium,” Medium, Feb. 10, 2023. https://medium.com/geekculture/ml-loss-functions-an-overview-14dcca261504

[20]TensorFlow, “tf.keras.losses.BinaryCrossentropy,” https://www.tensorflow.org/. https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.