

Event-Based Neural Network for Embedded Deployment

Representation-Model Co-Design and Event-Data Acquisition System

Enmin LIN

Promotor: Patrick Vandewalle

Co-promotor: Zijie Ning

Masterproef ingediend tot het behalen van
de graad van master of Science in de
industriële wetenschappen:
Elektronica-ICT

Academiejaar 2024 - 2025

©Copyright KU Leuven

Deze masterproef is een examendocument dat niet werd gecorrigeerd voor eventuele vastgestelde fouten.

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Campus De Nayer, Jan De Nayerlaan 5, B-2860 Sint-Katelijne-Waver, +32 15 31 69 44 of via e-mail iiw.denayer@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

As I put down my pen, marking the end of this thesis, I find myself looking back on the past year with deep gratitude. Completing this work is not just the conclusion of my master's journey at KU Leuven, but also a meaningful chapter in my life.

A year ago, I knew little about event cameras—just a spark of curiosity. I had no background in machine learning, no experience with event data, and certainly no idea I'd venture into such a challenging field. Back then, I was equal parts nervous and excited, driven by nothing more than a simple "let's try" attitude. Now, as I look back, the hesitations and uncertainties of those early days have quietly transformed into the growth, warmth, and friendships I gathered along the way.

First and foremost, I owe my deepest thanks to my promoter, Prof. Patrick Vandewalle. Thank you for your trust and guidance throughout this research. Whenever I felt lost, your patience and clear advice helped me find my way again. Having the opportunity to learn under your supervision during my master's has been a true privilege.

A special thank you goes to my co-promoter, Zijie Ning. From selecting the topic to refining experiments, from structuring the thesis to polishing the details—you've been there at every step. Every discussion with you was marked by genuine listening and respect for my ideas, never lacking in patience or encouragement. Your passion for knowledge, your rigor in research, and your dedication have deeply inspired me. The long hours and shared frustrations we went through have now become some of my most cherished memories.

To my colleagues at the EAVISE research group—thank you for your invaluable advice and support. Every brainstorming session was a joy, and every helping hand made this journey warmer.

To my family, my girlfriend Maro, and my friends—thank you for standing by me, whether in daily life, emotional struggles, or moments of doubt. Your unwavering support has been my anchor.

Finally, to everyone who has supported, encouraged, or simply been kind to me along this path—thank you. It's your generosity that has made my master's journey so rich and rewarding. The road ahead is long, but I hope we'll all carry forward with passion and courage.



*Copyright informatie:
studiewerk van een student in het kader van de
academische opleiding en examenbeoordeling.
Na deze beoordeling vond geen correctie plaats
van het studiewerk.*

Samenvatting

Diepgaand leren heeft het visueel waarnemingsvermogen bij autonoom rijden aanzienlijk verbeterd, maar conventionele camera's blijven beperkt door bewegingsonscherpte, een beperkt dynamisch bereik en hoge latentie. Gebeurteniscamera's, geïnspireerd door biologische visie, overwinnen deze beperkingen door asynchroon veranderingen in pixelhelderheid te detecteren. Dit mechanisme levert datastromen die schaars, temporeel nauwkeurig en zeer efficiënt zijn voor embedded toepassingen. De inherente schaarste en het gebrek aan intensiteitsinformatie in gebeurtenisgegevens bemoeilijken echter het extraheren van onderscheidende kenmerken en vormen uitdagingen voor de training van neurale netwerken. Daarnaast lijden gangbare datasets zoals CIFAR10-DVS aan een niet-uniforme verdeling van gebeurtenissen door het gebruik van lineaire of polygonaal gevormde bewegingstrajecten tijdens de opname, wat leidt tot periodieke artefacten.

Om deze beperkingen te overwinnen, evalueert dit proefschrift systematisch gebeurtenisrepresentaties—waaronder gestapelde afbeeldingen, voxelrasters en tijdoppervlakken—met standaard CNNs om optimale architecturen voor embedded implementatie te identificeren. Door gebruik te maken van transfer learning met vooraf getrainde ImageNet-modellen wordt de nauwkeurigheid onder beperkte dataomstandigheden aanzienlijk verbeterd. Experimentele resultaten tonen aan dat volledige fine-tuning van het hele netwerk noodzakelijk is om zich aan te passen aan de unieke eigenschappen van gebeurtenisgebaseerde gegevens. Het voorgestelde model integreert dropout, L2-regularisatie en een meerfasig leersnelheidsschema om de trainingsstabiliteit en generalisatie te bevorderen. Met deze co-designbenadering behaalt EfficientNetB0 een nauwkeurigheid van 81,05% op CIFAR10-DVS bij verwerking van gebeurtenissen binnen één temporeel venster.

Voor implementatie identificeert inspectie met het Xilinx Vitis AI-platform TensorFlow's MobileNetV2 als de optimale keuze dankzij volledige compatibiliteit met DPU-hardwareversnellers. Hoewel post-training pruning op basis van globale kanaalrangschikking is onderzocht, is de effectiviteit beperkt door het inherent gebrek aan redundantie in het model. Daarentegen compenseert quantization-aware training (QAT) succesvol het nauwkeurigheidsverlies tijdens INT8-kwantisatie, waarmee real-time prestaties (2,8 ms/frame) bij een ultralaag energieverbruik (1,77 mW) worden bereikt op het KV260 embedded platform.

Gemotiveerd door de beperkingen van bestaande datasets wordt een nieuw acquisitieplatform voorgesteld, waarbij gebruik wordt gemaakt van een Dobot-robotarm voor continue

cirkelvormige beweging geïnspireerd op biologische oogbewegingen. Deze methode genereert uniforme gebeurtenisverdelingen en elimineert periodieke artefacten die aanwezig waren in eerdere datasets, wat robuustere training en evaluatie van neurale netwerken op gebeurtenisgegevens ondersteunt.

Summary

Deep learning has substantially advanced visual perception in autonomous driving, yet conventional cameras remain limited by motion blur, restricted dynamic range, and high latency. Event cameras, inspired by biological vision, overcome these constraints by asynchronously detecting changes in pixel brightness. This mechanism produces data streams that are sparse, temporally precise, and highly efficient for embedded applications. However, the inherent sparsity and lack of intensity information in event data complicate the extraction of discriminative features and pose challenges for neural network training. Additionally, commonly used datasets such as CIFAR10-DVS suffer from non-uniform event distributions due to the use of linear or polygonal motion trajectories during recording, resulting in periodic artifacts.

To address these limitations, this thesis systematically evaluates event data representations—including stacked images, voxel grids, and time surfaces—with standard CNNs to identify optimal architectures for embedded deployment. Leveraging transfer learning from ImageNet-pretrained models significantly improves accuracy under limited data conditions, where experimental results demonstrate the necessity of full network fine-tuning to adapt to the unique characteristics of event-based data. The proposed model incorporates dropout, L2 regularization, and a multi-phase learning rate schedule to enhance training stability and generalization. Through this co-design approach, EfficientNetB0 achieves 81.05% accuracy on CIFAR10-DVS when processing events within a unified temporal window.

For deployment, inspection using the Xilinx Vitis AI platform identifies TensorFlow’s MobileNetV2 as the optimal choice due to its full compatibility with DPU hardware accelerators. While post-training pruning based on global channel ranking is explored, its effectiveness is limited by the model’s inherent lack of redundancy. In contrast, quantization-aware training (QAT) successfully compensates for accuracy loss during INT8 quantization, achieving real-time performance (2.8 ms/frame) at ultra-low power consumption (1.77 mW) on the KV260 embedded platform.

Motivated by the limitations of existing datasets, a novel acquisition platform utilizing a Dobot robotic arm is proposed, enabling continuous circular motion inspired by biological eye movements. This method generates uniform event distributions and eliminates periodic artifacts present in earlier datasets, supporting more robust neural network training and evaluation.

Keywords: Event camera, CIFAR10-DVS, CNN, Transfer learning, Embedded deployment,

MobileNetV2, QAT.

Contents

1	Introduction	4
1.1	Motivation from Autonomous Driving Scenarios	4
1.2	Event Cameras: Principles, Advantages, and Applications	4
1.3	Current Challenges in Event-Based Neural Processing	5
1.4	Thesis Contributions	6
2	Related Work	7
2.1	Event Camera Mechanism and Characteristics	7
2.2	Event Data Representations	9
2.3	Neural Network Architectures for Event Data Classification	11
2.3.1	SNNs	11
2.3.2	GNNs	11
2.3.3	Transformer Architectures	12
2.3.4	CNNs	12
2.3.5	Comparison of Event Processing Architectures for Embedded De- ployment	13
2.4	Transfer Learning in Event-Based Vision	13
2.5	Converting Static Image Datasets to Event-Based Datasets	14
2.6	Summary	16
3	Training and Evaluation of Neural Networks on CIFAR10-DVS	18
3.1	Methods	18
3.1.1	Event Representation	18
3.1.2	Model Selection	18
3.1.3	Benchmark Dataset and Model Initialization	20
3.1.4	Hyperparameter Search Methodology	20
3.2	Results and Analysis	22

3.2.1	Search results based on data representation and model architecture	22
3.2.2	Search results based on time stack layers and frozen layers	24
3.2.3	Final Best Model	25
3.2.4	Temporal Window Dependence in CIFAR10-DVS Classification	28
3.3	Summary	29
4	Neural Network Deployment Strategies	31
4.1	Inspection	32
4.2	Pruning	33
4.2.1	Pruning Fundamentals	33
4.2.2	Unstructured and Structured Pruning	34
4.2.3	Pruning in the Vitis AI Deployment Flow	35
4.3	Quantization and Fine-tuning	36
4.3.1	Post-training Quantization	37
4.3.2	Quantization-aware Training (QAT fine-tuning)	38
4.4	Compilation	39
4.5	Deployment on KV260	39
4.5.1	Deployment Throughput and Efficiency on KV260	39
4.5.2	Temporal Window Size and Per-Class Accuracy	40
4.6	Summary	41
5	Event data recording platform system based on circular motion tracking	43
5.1	Motivation from CIFAR10-DVS	43
5.2	Biological Interpretability of Rotational Motion	46
5.3	Recording Platform Design	47
5.3.1	System Setup	47
5.3.2	Time Synchronization	48
5.3.3	ROI Tracking	49
5.4	Conclusion	50
6	Conclusion and future work	52
6.1	End-to-End Event Vision System Framework and Overall Innovations	52
6.2	Key Experimental Findings and Comparative Analysis	52
6.3	Limitations and Future work	54
A	Code Repository	61

Acronyms

.h5 Hierarchical Data Format version 5. 31

ADAS advanced driver-assistance system. 4

API application programming interface. 47

AprilTag visual fiducial marker for pose estimation. 48, 49

ASIC application-specific integrated circuit. 13, 40

CIFAR-10 Canadian Institute For Advanced Research 10-class dataset. 15, 26

CIFAR10-DVS CIFAR-10 Dynamic Vision Sensor dataset. 6, 9, 13, 14, 15, 16, 18, 20, 22, 27, 28, 29, 40, 41, 43, 44, 45, 50, 51, 53, 54

CNN convolutional neural network. 6, 9, 10, 11, 12, 13, 14, 16, 18, 20, 22, 23, 52

ConstantLR constant learning rate. 26

CosineAnnealingLR cosine annealing learning rate. 26

DAVIS346 Dynamic and Active-pixel Vision Sensor with 346×260 resolution. 47, 48, 49

DPU Deep Processing Unit. 32, 33, 34, 39, 41

Dropout dropout regularization. 25, 26, 30, 52, 53

DVS dynamic vision sensor. 4, 7, 9, 14, 15, 20, 52

EfficientNetB0 EfficientNet-B0 network. 18, 19, 22, 23, 24, 25, 26, 29, 33, 53, 54

FFT fast Fourier transform. 44, 50, 51

FP32 32-bit floating-point. 28, 37

FPGA field-programmable gate array. 13, 40, 52, 53

GET Group Event Transformer. 12, 53

GFLOPs giga floating-point operations per second. 19, 39, 40

GlobalAvgPool global average pooling. 19, 27

GNN graph neural network. 10, 11, 12, 13

ImageNet ImageNet Large Scale Visual Recognition Challenge dataset. 11, 13, 14, 15, 52, 54

INT8 8-bit integer. 36, 37, 38, 39, 40

KV260 AMD-Xilinx Kria KV260 Edge AI Starter Kit. 6, 34, 39, 40, 41, 55

L2 L2 regularization. 25, 30, 53

LCD liquid crystal display. 14, 15, 16

LinearLR linear learning rate. 25

LR learning rate. 25, 26, 52

MBConv Mobile Inverted Bottleneck Convolution. 19

MNIST Modified National Institute of Standards and Technology database. 14

MNIST-DVS Modified National Institute of Standards and Technology database - Dynamic Vision Sensor. 14, 15, 16

MobileNetV2 MobileNet-V2 network. 25, 27, 30, 32, 33, 36, 40, 41, 53, 54, 55

NCHW [batch, channel, height, width] data format. 33

NHWC [batch, height, width, channel] data format. 33

N-MNIST Neuromorphic MNIST event dataset. 16

N-Cars Neuromorphic Cars event dataset. 15

N-ImageNet Neuromorphic ImageNet event dataset. 15, 16, 54

N-Caltech101 Neuromorphic Caltech101 event dataset. 10, 15, 16

ONNX Open Neural Network Exchange. 13

PDAF phase detection auto focus. 47

PTQ post-training quantization. 36, 37, 53

QAT quantization-aware training. 31, 36, 37, 38, 39, 52, 53

RCLS repeated closed-loop smooth. 15, 16, 43

ReLU rectified linear unit. 19, 27, 33, 38, 55

ResNet18 Residual Network 18-layer. 18, 19, 22, 23, 24, 29

RGB red, green, blue. 48

ROI region of interest. 49, 50, 61

SAE surface of active events. 10, 23

SGD stochastic gradient descent. 26

SiLU sigmoid-weighted linear unit. 32, 33, 41, 53

SNN spiking neural network. 10, 11, 13

Swish Swish activation function. 19, 33, 53

Unix Unix time, number of seconds since 1970-01-01 00:00:00 UTC. 48

VGG16 Visual Geometry Group 16-layer network. 18, 19, 22, 23, 24, 29

Vitis AI Xilinx Vitis AI toolchain. 31, 32, 33, 35, 36, 37, 38, 39, 41, 54

wandb Weights & Biases. 20

Chapter 1

Introduction

1.1 Motivation from Autonomous Driving Scenarios

Deep learning with frame-based cameras has become a cornerstone of modern computer vision systems in autonomous driving.[1] Leveraging monocular or stereo cameras, tasks such as object detection, lane tracking, and semantic segmentation have achieved remarkable accuracy. These advances are largely driven by the success of deep learning, which enables the extraction of meaningful visual features from image sequences. As a result, frame-based vision systems have been widely deployed in real-world applications and remain a key component in commercial advanced driver-assistance system (ADAS).

Despite these achievements, traditional cameras operating at fixed frame rates face fundamental limitations. Their performance is often constrained by motion blur, especially in fast-moving scenes, and they struggle in challenging lighting conditions due to limited dynamic range. These issues introduce perceptual delays and compromise reliability in scenarios such as tunnels, glare, or nighttime driving. In safety-critical applications like autonomous driving, such delays can hinder timely object detection and decision-making, potentially leading to hazardous outcomes. Furthermore, the need to capture and process full image frames at regular intervals leads to high data throughput and elevated energy demands, which can be problematic for embedded systems with limited resources.

These challenges highlight the need for faster and more adaptive vision systems that can respond effectively to dynamic environments, while maintaining reasonable efficiency in computation and power usage.

1.2 Event Cameras: Principles, Advantages, and Applications

Event cameras, also known as dynamic vision sensor (DVS), are bio-inspired imaging devices that operate fundamentally differently from traditional frame-based cameras. Instead of capturing full images at fixed intervals, an event camera outputs a stream of asynchronous

events triggered independently by each pixel when it detects a change in logarithmic brightness.[2] This sensing mechanism mimics the functionality of the human retina and enables the camera to react immediately to local brightness variations in the scene.

The resulting data is inherently sparse and temporally precise. Each event encodes the pixel location, timestamp, and polarity of the brightness change, allowing for microsecond-scale temporal resolution. Unlike conventional cameras, which suffer from motion blur and redundant frame data, event cameras produce a low-latency stream that directly reflects dynamic changes in the environment.

Event cameras offer several key advantages. Their high dynamic range (often exceeding 120 dB) enables robust performance in scenes with extreme lighting contrast, such as entering or exiting tunnels. Due to their asynchronous operation, they avoid motion blur even in high speed scenarios. Additionally, since only changing pixels generate output, event cameras consume significantly less power and produce lower data rates compared to standard image sensors. This characteristic makes them suitable for embedded and edge computing applications.[3]

In recent years, event cameras have been successfully applied in autonomous driving systems. For example, they have been used to predict vehicle steering angles under difficult conditions such as nighttime and glare, outperforming traditional camera-based models in terms of both accuracy and robustness [4]. Moreover, hybrid perception systems combining event and frame data have achieved comparable latency to ultra-high-frame-rate cameras while maintaining low bandwidth usage [5]. These applications demonstrate the potential of event cameras to enhance real-time perception in challenging automotive scenarios.

1.3 Current Challenges in Event-Based Neural Processing

Despite the growing interest in event-based vision, several challenges remain in applying deep neural networks effectively to event data. First, extracting discriminative features from asynchronous and sparse event streams remains non-trivial. Unlike dense frame-based images, event data lacks absolute intensity and contextual information, which introduces new challenges in designing feature extraction strategies and adapting existing network architectures. As a result, classification accuracy often lags behind that of well-established frame-based systems, particularly in low-data or highly variable conditions. [6]

Second, the process of recording event-based datasets is not yet standardized. Many existing datasets are derived from static images displayed on monitors with simulated motion, where either the event camera moves or the image is moved across the screen. These inconsistent data collection methods introduce variations in spatiotemporal event distribution, which can significantly affect the training and evaluation of learning models. In addition, the choice of motion trajectory - linear, triangular, or circular is often made arbitrarily without theoretical justification, leading to further variation across datasets. The absence of a principled data acquisition methodology or task - specific motion design limits reproducibility and hinders rig-

orous comparison. Furthermore, the lack of ground truth alignment across datasets impedes systematic benchmarking and comparison.

Third, many event-based learning models and data representation methods are not inherently designed for efficient deployment on embedded hardware [7]. Models and representations developed primarily for research or server platforms often require significant adaptation before they can run on resource-constrained edge devices. For example, representations with unnecessary redundancy or high computational complexity can create bottlenecks on embedded hardware. Addressing these issues requires careful alignment of data representation and model architecture with the actual deployment environment.

1.4 Thesis Contributions

To address the aforementioned challenges in a systematic and integrated manner, this thesis proposes a comprehensive framework that spans the entire pipeline - from event data acquisition to neural network training and real-world embedded deployment. Rather than treating each subproblem in isolation, we emphasize the co-design of data representation, model architecture, and hardware constraints to achieve practical, high-performance event-based systems.

The specific contributions of this thesis are as follows:

- Developed a co-design methodology for selecting data representations and neural network architectures optimized for embedded deployment, balancing model size and accuracy.
- Applied transfer learning to lightweight convolutional neural network (CNN), reaching 81.05% classification accuracy on CIFAR-10 Dynamic Vision Sensor dataset (CIFAR10-DVS).
- Implemented and validated event-based neural networks on an AMD-Xilinx Kria KV260 Edge AI Starter Kit (KV260), providing quantitative analysis of latency, resource utilization, and accuracy to confirm real-time deployment feasibility.
- Analyzed the limitations of current event-camera datasets that use screen-based playback, and identified the unreasonable generation of events caused by existing motion trajectories. Proposed a circular-motion-based recording platform to ensure temporal consistency and spatial clarity in event streams.

Chapter 2

Related Work

This chapter reviews the main research directions in event-based vision, including the principles of event cameras, methods for event data representation, neural network models for event processing, embedded deployment strategies, transfer learning with limited data, and the creation of static event datasets.

2.1 Event Camera Mechanism and Characteristics

Inspired by the human retina, event cameras, also known as DVS, operate fundamentally differently from traditional frame-based image sensors [2]. Rather than capturing full image frames at fixed intervals, they asynchronously detect local changes in brightness at the pixel level, mimicking the retina's ability to respond only to significant variations in light intensity.

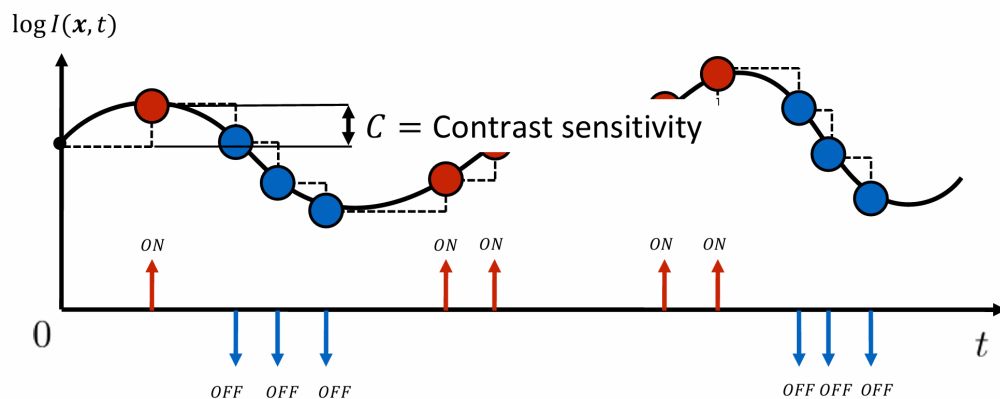


Figure 2.1: A simple demonstration of the mechanism by which event cameras generate event data per pixel. Illustration adapted from UZH tutorial [8]

Each pixel independently monitors, as illustrated in Figure 2.1, the logarithmic intensity of incoming light. When the change in log-intensity exceeds a predefined threshold, the pixel triggers an event. This operation is implemented using a photoreceptor circuit with

logarithmic response, followed by a capacitive differencing stage that compares the current signal to a reference. The polarity p of an event is determined according to Equation 2.1, which defines the thresholding condition based on contrast sensitivity, where u denotes the spatial coordinate of the event. If the condition is satisfied, the pixel emits either an ON or OFF event. Each event is represented as a tuple (t, x, y, p) , where t is the timestamp of the event, (x, y) indicates the spatial location of the pixel, and p denotes the polarity, indicating whether the brightness increased (ON) or decreased (OFF). Events are transmitted immediately via an asynchronous address-event representation (AER) bus, which avoids the need to wait for an entire frame capture, as shown in Figure 2.2.

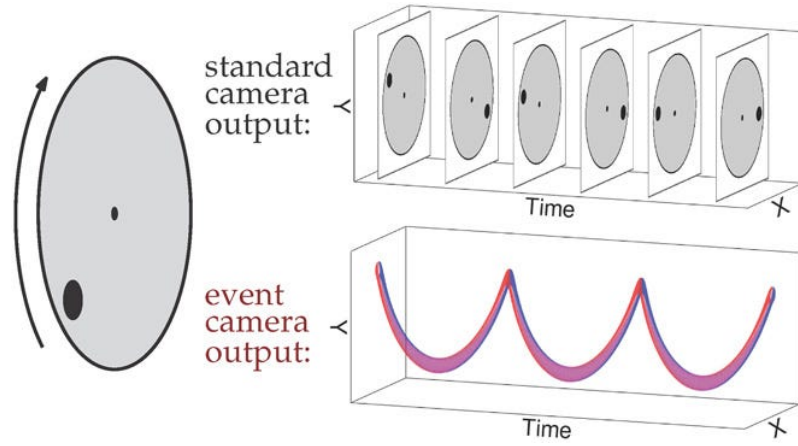


Figure 2.2: Comparison between a standard frame-based camera and an event camera during rotational motion. The rotating disk with one dark markers is observed. The top shows the frame camera output: discrete grayscale frames captured at fixed intervals. The bottom shows the event camera output: a continuous stream of brightness changes, where red and blue denote ON and OFF events respectively. The event stream provides high temporal resolution and captures motion without redundant static information. Illustration adapted from UZH tutorial [8]

$$p = \begin{cases} +1, & L(u, t) - L(u, t - \Delta t) \geq C \\ -1, & L(u, t) - L(u, t - \Delta t) \leq -C \end{cases} \quad (2.1)$$

This pixel-level, asynchronous design leads to several key characteristics:

- The use of logarithmic response photoreceptors enables adaptation to a wide range of lighting conditions and allows the sensor to achieve a high dynamic range (HDR) typically exceeding 120 dB.
- Events are generated only when brightness changes. This results in sparse output and reduces data redundancy compared to conventional image sensors.
- Each event is emitted with microsecond-scale latency (as low as 15 microseconds),

providing real-time responsiveness in dynamic scenes without the constraints of fixed frame rates.

- The overall system consumes significantly less power, such as 23 *mW* for the original 128×128 DVS [2]. This makes it well-suited for embedded and energy-constrained applications.

Since each pixel operates independently, the sensor supports truly asynchronous event triggering and achieves precise temporal resolution without the need for global shutter synchronization. These characteristics make event cameras particularly effective for vision tasks that require high-speed perception, high dynamic range, and low-power operation.

2.2 Event Data Representations

Event data generated by event cameras have sparse and spatio-temporal asynchronous characteristics. In order to better adapt to existing computer vision tasks, researchers have proposed a variety of representations, which mainly include image, surface, voxel, graph, impulse and learning-based representation. As shown in Figure 2.3a, this example visualizes event data collected from the CIFAR10-DVS [9].

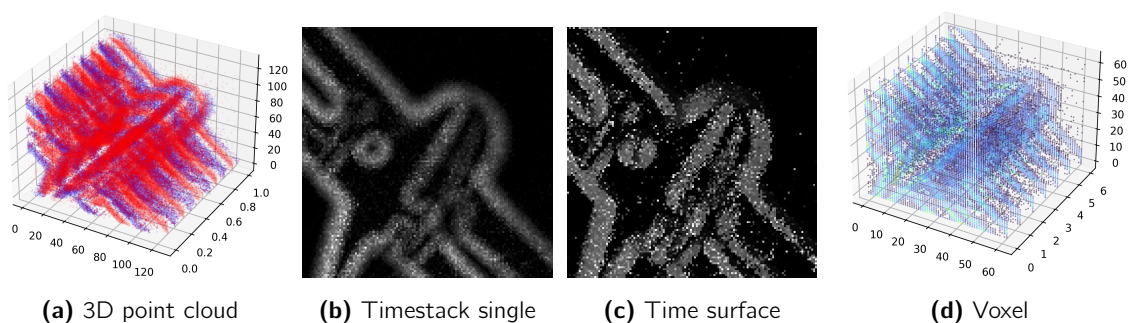


Figure 2.3: Comparison of event data representations: (a) Events are shown in (x, y, t) space, with red and blue dots indicating positive and negative polarity. (b) Events within a time window are stacked into an image, with pixel intensity reflecting event count. (c) Pixel brightness encodes the timestamp of the most recent event at each location. (d) Events are accumulated into 3D grid cells across spatial and temporal axes, with color indicating event density.

Image representation: This approach converts event streams into synchronous image-like frames, enabling compatibility with CNNs. Common strategies include event count and stacking events by polarity or timestamp. Maqueda et al. [4] proposed a polarity-based stacking method to generate two-channel images for real-time driving tasks. A time-sensitive tensor representation is employed, where events within a time window are accumulated separately for each polarity channel to form a three-dimensional tensor, as proposed in [10]. An illustration is shown in Figure 2.3b.

Surface representation: Temporal surfaces store the most recent event's timestamp at each pixel, preserving local temporal dynamics. To model the temporal proximity of past events, a time decay function is applied to construct a continuous-valued surface. The exponential time surface is defined as Equation 2.2. The surface of active events (SAE) [11] is a representative method that provides temporally informative maps used in motion analysis. An example is shown in Figure 2.3c.

Voxel representation: This method partitions the spatio-temporal domain into discrete volumes to accumulate events across time bins, allowing the use of 3D convolution. It offers a compromise between spatial resolution and temporal fidelity, as demonstrated in multiple works [12], [13]. See Figure 2.3d for an example.

Graph representation: Events are represented as nodes in a structured graph with spatial or temporal edges. Bi et al. [14] proposed a residual graph neural network (GNN) to reduce redundancy, while Deng et al. [15] introduced a voxel-based graph model that achieves 67.8% accuracy on the Neuromorphic Caltech101 event dataset (N-Caltech101) with fewer parameters compared to CNN baselines.

Impulse representation: This format models event data as discrete temporal pulses, supporting neural models that encode time directly. It aligns with the principles of spiking neural networks (SNNs) and has been used for neuromorphic inference tasks [16],[17].

Learning-based representation: Instead of designing representations manually, this approach learns data-driven embeddings directly from raw events. Gehrig et al. [18] employed a multilayer perceptron to learn coordinate-based features. Matrix-LSTM [19] integrates event sequences using a grid of temporal cells in a fully differentiable structure.

$$S(x, y, t, p) = \exp \left(-\frac{t - T_p(x, y)}{\tau} \right) \quad (2.2)$$

Each of these representations imposes different structural, temporal, and sparsity characteristics on the input data, which in turn influence the design and performance of downstream neural networks. For instance, dense representations such as stacked frames or voxel grids are naturally suited for convolutional architectures, while sparse or asynchronous formats align better with graph-based or neuromorphic models. Therefore, the choice of representation is closely coupled with the selection of network architecture. In the following section, we examine how event-based neural networks have been developed or adapted to accommodate the unique properties of different representations and to meet the practical constraints of embedded deployment.

2.3 Neural Network Architectures for Event Data Classification

In the evolution of computer vision, neural networks have progressively supplanted traditional image processing methods through their unique hierarchical feature extraction capabilities. This technological advancement is marked by three pivotal breakthroughs: LeCun et al.'s LeNet-5 [20] first demonstrated the effectiveness of CNNs in handwritten character recognition; Krizhevsky et al.'s AlexNet [21] achieved groundbreaking results in the ImageNet Large Scale Visual Recognition Challenge dataset (ImageNet) competition through deep architecture and ReLU activation functions; He et al.'s ResNet [22] addressed the challenge of training deep networks via residual connections. These landmark works collectively established neural networks' dominance in visual tasks, attributable to three core advantages: 1) translation-invariant feature extraction through convolutional kernels, 2) end-to-end training paradigm, and 3) high compatibility with GPU computing architectures. However, when processing the asynchronous sparse data generated by event cameras, these conventional vision network architectures require specific modifications and optimizations to fully realize their performance potential.

2.3.1 SNNs

SNN implement a biologically inspired computation paradigm where information is processed through discrete spike events rather than continuous activations. In SNNs, each neuron maintains an internal membrane potential that integrates incoming spikes over time. When this potential crosses a predefined threshold, the neuron fires an output spike and resets its potential. This event-driven operation provides a natural compatibility with event camera outputs, as both systems rely on asynchronous, sparse data representation. The HFirst architecture [23] demonstrates this compatibility through a hierarchical Winner-Take-All mechanism, achieving real-time performance in constrained recognition tasks while maintaining low power consumption. The architecture's spike-based computation aligns well with the event camera's data generation principles, as both operate on discrete, asynchronous events rather than frame-based representations.

2.3.2 GNNs

GNNs process event data by constructing dynamic graph structures where each event corresponds to a node in the graph. Connections between nodes are established based on spatial proximity or temporal co-occurrence, preserving the inherent sparsity of event streams while enabling structured computation through message passing between connected nodes. The Bi-Graph CNN architecture [24] implements this approach through bidirectional graph convolution operations, effectively capturing object motion patterns while maintaining computational efficiency. The RG-CNN variant [14] further improves performance through residual connections that enhance information flow through the network, particularly beneficial under

conditions of sparse event data. These graph-based approaches excel at extracting localized spatiotemporal features from event streams, making them particularly suitable for analyzing dynamic scenes where object motion patterns are critical for recognition.

2.3.3 Transformer Architectures

Transformer architectures have been adapted for event-based vision by treating event streams as temporal sequences and applying self-attention mechanisms to model relationships between events. The Group Event Transformer (GET) [25] represents a significant advancement in this direction, organizing events into spatiotemporal groups based on their coordinates, timestamps, and polarity before applying attention operations. The architecture employs a two-stage attention mechanism that first processes local event patterns within each group, then models global relationships between different groups. This hierarchical approach achieves state-of-the-art performance on standard benchmarks by effectively capturing both fine-grained details and long-range dependencies in the event stream. The attention mechanisms provide superior modeling capacity compared to local operations in CNNs or GNNs, but this comes with substantial computational overhead that grows quadratically with input size.

2.3.4 CNNs

CNN have been widely explored for event camera data classification due to their outstanding performance in traditional image processing tasks. HATS (Histograms of Averaged Time Surfaces) [6] is a representative CNN-based approach for event data processing. Its core idea involves transforming event streams into Local Memory Time Surfaces and aggregating them into fixed-dimensional feature representations through histograms, which are then fed into a linear classifier for classification. The strength of HATS lies in its computational efficiency, effective utilization of spatiotemporal locality in event data, and reduced redundant computations through shared memory units, making it suitable for real-time embedded systems. However, HATS still relies on handcrafted feature extraction rather than end-to-end learning, which may limit its generalization capability in complex tasks.

Beyond HATS, several well-established CNN architectures, such as ResNet [22], VGG [26], MobileNetV2 [27], and EfficientNetB0 [28] can also be adapted for event data classification. These networks automatically learn hierarchical features through stacked convolutional layers, offering greater adaptability compared to manually designed features. For instance, ResNet's residual connections mitigate gradient vanishing in deep networks, while MobileNetV2's depthwise separable convolutions significantly reduce computational overhead, making it more suitable for resourceconstrained embedded platforms.

2.3.5 Comparison of Event Processing Architectures for Embedded Deployment

While each architecture offers unique advantages for event processing, all face significant barriers to practical deployment on resource-constrained embedded platforms. SNNs require specialized neuromorphic processors like Intel Loihi or IBM TrueNorth [29] that are not commonly supported in standard embedded toolchains, limiting their practical adoption. GNNs suffer from unpredictable execution patterns due to their dynamic graph structures [30], leading to inefficient memory access patterns that cause frequent cache misses and memory bottlenecks. The irregular data access patterns in GNNs make it difficult to achieve efficient hardware utilization on standard embedded processors. Transformer architectures [31] present perhaps the most severe challenges due to their quadratic memory growth with input size and computationally intensive attention operations, making real-time performance difficult to achieve without compromising model accuracy or temporal resolution. These fundamental limitations across all three approaches motivate the investigation of alternative architectures that can better balance computational efficiency with effective event processing capabilities for embedded deployment.

The advantages of CNNs in embedded deployment are reflected in three key aspects. First, their highly regular computation patterns facilitate parallel optimization using GPUs or dedicated accelerators (e.g., field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs)). Second, mature model compression techniques, such as quantization, pruning, and knowledge distillation can further reduce storage and computational costs. Third, mainstream deep learning frameworks (e.g., TensorFlow Lite and Open Neural Network Exchange (ONNX) Runtime) provide robust support for CNNs, simplifying practical deployment.

2.4 Transfer Learning in Event-Based Vision

In visual classification tasks, deep learning models have achieved significant progress, particularly CNNs trained on large-scale supervised datasets such as ImageNet, which demonstrate strong feature extraction capabilities across various tasks. However, traditional deep learning methods heavily rely on large-scale labeled samples, an assumption that often does not hold in real-world scenarios. Taking CIFAR10-DVS[9] as an example, this event camera dataset contains only around 10,000 images with 10 categories, making directly trained deep models prone to overfitting and poor generalization. To address performance degradation under small-sample conditions, transfer learning provides an effective solution. By leveraging knowledge learned from a source task (e.g., ImageNet classification) and transferring it to a target task, model efficiency and accuracy in the target domain can be significantly improved, especially in data-scarce scenarios.

In recent years, the application of transfer learning in visual recognition has deepened, achiev-

ing systematic progress in few-shot learning and lightweight model adaptation. Huh [32] empirically demonstrated that CNN features pre-trained on ImageNet exhibit strong generalizability, with high-level semantic features being transferable across various downstream tasks and domains. This finding provides a theoretical basis for deploying complex networks on small- to medium-scale datasets. In practice, fine-tuning has become one of the most common transfer strategies. By freezing shallow layers of a pre-trained model to retain low-level feature extraction (e.g., edges and textures) and adjusting only higher-level modules, parameter redundancy can be reduced while maintaining or even improving transfer efficiency. In the field of event vision, multiple studies have validated the effectiveness of transfer learning strategies. Yang et al. [33] proposed an Event Camera Data Pre-training framework, which introduces a self-supervised pre-training scheme specifically designed for event data. This method combines event image augmentation, conditional masking, and cross-modal contrastive constraints to learn stable visual representations from event streams without manual annotation. In multiple downstream tasks, this approach demonstrates strong transferability, particularly on the CIFAR10-DVS, achieving a classification accuracy of 78%, which significantly outperforms baseline models trained without pre-training. This research confirms the feasibility of event data pre-training in enhancing classification performance and provides theoretical and methodological support for deploying event vision systems in few-shot, high-dynamic scenarios.

2.5 Converting Static Image Datasets to Event-Based Datasets

The evolution of event camera technology has facilitated the creation of numerous specialized datasets for image classification tasks. However, direct acquisition of native event data remains challenging, requiring dedicated DVS hardware along with significant temporal and financial resources. Consequently, many widely-used event datasets have been generated by converting conventional frame-based image datasets using physical event cameras, providing a more practical alternative for researchers.

The Modified National Institute of Standards and Technology database - Dynamic Vision Sensor (MNIST-DVS) [34] dataset represents the first event-based dataset created by physically recording static images displayed on a liquid crystal display (LCD) screen. In this pioneering work, each Modified National Institute of Standards and Technology database (MNIST) handwritten digit was displayed on a high-resolution LCD monitor and moved along controlled, slow-speed random trajectories to generate event streams. A key technical aspect involved the deliberate resolution mismatch between the high-resolution LCD display and the lower-resolution DVS (128×128 pixels), creating a many to one pixel mapping that effectively provided optical smoothing of the event data. During data collection, the researchers identified significant artifacts caused by the LCD's inherent refresh rate mechanism, which introduced periodic patterns unrelated to the actual image motion. To address this, the authors developed an innovative timestamp randomization technique, as detailed in their

supplementary documentation, that preserves the essential temporal ordering of events while redistributing their absolute timestamps. This approach successfully eliminated refresh-rate-induced periodicity without compromising the dataset's temporal characteristics, establishing an important methodological precedent for subsequent event dataset generation.

The CIFAR10-DVS [9] dataset was constructed by selecting 1,000 images per category from the original Canadian Institute For Advanced Research 10-class dataset (CIFAR-10) dataset and converting them into event streams through physical recording. Each image was displayed on an LCD monitor and moved along a predefined repeated closed-loop smooth (RCLS) trajectory at a constant velocity of 200 pixels per second. The RCLS path followed a 45-degree rhombus pattern, with each image completing six full cycles of this motion during recording. To mitigate artifacts caused by the monitor's refresh rate, the same timestamp randomization technique employed in MNIST-DVS was applied, preserving the temporal ordering of events while eliminating periodic noise components introduced by the display hardware. The recording setup utilized a DVS camera positioned at a fixed distance from the LCD screen.

The Neuromorphic ImageNet event dataset (N-ImageNet) [35] dataset was developed to address the limitations of existing event camera datasets, which either cover limited categories (e.g., Neuromorphic Cars event dataset (N-Cars)) or contain insufficient samples (e.g., N-Caltech101) for large-scale object recognition tasks. Designed to serve as the event vision counterpart to ImageNet's role in conventional computer vision, N-ImageNet employs a distinct data acquisition methodology where images remain stationary on a display screen while the event camera executes a square trajectory in counterclockwise rotation to capture event streams. Crucially, because the camera moves rather than the displayed images, this approach avoids artifacts caused by monitor refresh rates, eliminating the need for timestamp randomization techniques used in datasets like MNIST-DVS. The recording setup ensures comprehensive coverage of visual features through systematic camera movements, with the square path providing uniform sampling of orientation-dependent event patterns.

Orchard et al. [36] pointed out that in the process of converting static images into spiking neuromorphic datasets, asynchronous event sensors can be activated by motion-induced brightness changes. This process can be modeled using the optical flow equation, which expresses the temporal derivative of image intensity as a function of spatial gradients and motion. Event cameras are insensitive to image gradients aligned with the direction of motion. Under translational movement, their responses become dependent on scene depth. In contrast, pure rotational motion eliminates this depth dependency, enabling uniform event activation across the image regardless of object distance. This mechanism is particularly suitable for static-to-event data conversion tasks, as it facilitates consistent visual stimulation across the image.

They further emphasized that rotational motion has not only technical advantages but also biological plausibility. During fixation on a static object, the human eye exhibits small involuntary fixational eye movements, including microsaccades, tremor, and drift, which maintain continuous image refresh on the retina. Among these, microsaccades involve small transla-

tional displacements, while tremor and drift are characterized by slow, involuntary rotations components. These rotational components trends induce sustained brightness changes on the retina without relying on depth variations in the scene [37]. From a signal perspective, smooth rotational trajectories produce frequency components concentrated at the actual motion frequency (e.g., 10 Hz) and at lower frequencies determined by the recording duration (e.g., approximately 3.3 Hz), effectively avoiding artifacts caused by the refresh rate of screen-based stimulation methods. However, despite the perceptual advantages of pure rotational motion, the Neuromorphic MNIST event dataset (N-MNIST) and N-Caltech101 datasets were collected using a three-phase microsaccade trajectory forming an isosceles triangle, which reintroduces short-duration translational components into the motion pattern.

Table 2.1 Summary of Trajectory Designs in Major Static-to-Event Datasets

Dataset	Moving Object	Trajectory Type	Description
MNIST-DVS	Image	Random slow path	Image moves on LCD along slow, controlled random trajectories
CIFAR10-DVS	Image	RCLS (diamond, closed-loop)	Image moves on LCD along a 45-degree diamond trajectory, repeating six cycles
N-ImageNet	Camera	Square, closed-loop	Camera moves along a square trajectory in front of a stationary image (counterclockwise)
N-MNIST	Camera	Microsaccade (triangle)	Camera follows a three-phase microsaccade forming an isosceles triangle

2.6 Summary

Based on the comparative analysis above, this work adopts CNNs in combination with image-based event representations as the primary architectural framework. Building on this foundation, Chapter 3 focuses on identifying an optimal co-design strategy that jointly selects event representations and CNN architectures. The proposed system will be evaluated in terms of classification performance and resource efficiency, culminating in a complete deployment pipeline targeting embedded platforms.

Transfer learning is adopted as a core strategy to enhance event-based classification, particularly for small-scale datasets where training deep models from scratch is insufficient. Transfer learning enables effective reuse of pre-trained visual features, providing a solid initialization that improves convergence and generalization. Among the available approaches, fine-tuning offers a flexible mechanism for adaptation, allowing selective adjustment of network layers.

This is especially promising for event-based vision, where the optimal layer-freezing scheme remains an open research question due to the inherent differences between event and frame-based data. By systematically exploring these design choices, such as which layers to freeze, this work aims to extend fine-tuning practices to the event domain, ultimately enabling efficient and accurate deployment of event vision systems on embedded platforms.

As shown in Table 2.1, these observations collectively underscore that across various static-to-event dataset generation pipelines, there is no unified convention regarding the choice of motion target-whether to move the stimulus image or the sensor itself-or the design of motion trajectories. The absence of standardization in these factors leads to inconsistencies in the spatial structure, density, and edge quality of the recorded events, which in turn can impact downstream model training, evaluation, and generalization. This issue will be further examined in Chapter 5, where we analyze how different motion strategies affect edge preservation and event generation, and present a dedicated platform for capturing static image event datasets with precisely controlled, biologically inspired motion trajectories.

Chapter 3

Training and Evaluation of Neural Networks on CIFAR10-DVS

3.1 Methods

This section describes the approach used to develop and evaluate neural networks for event-based classification on the CIFAR10-DVS dataset. It outlines the process of selecting and preparing event data representations, explains the rationale for choosing specific neural network architectures, and details the procedures for model initialization and transfer learning. The methodology for conducting systematic hyperparameter search and experiment tracking is also introduced, providing a clear foundation for the experiments that follow.

3.1.1 Event Representation

The representation methods selected in this work include image-based formats and voxel-based formats, which discretize the spatiotemporal event stream into 3D grids where each voxel accumulates the number of events within its bin. These representations apply minimal preprocessing to the raw event streams while retaining sufficient spatial and temporal information for downstream neural network processing. Their structural simplicity ensures compatibility with standard convolutional architectures and enables efficient data throughput in embedded systems. In particular, the low computational overhead of these representations for high data throughput during preprocessing, which is critical for maintaining real-time constraints during embedded deployment.

3.1.2 Model Selection

The three CNN architectures selected in this study—Visual Geometry Group 16-layer network (VGG16), Residual Network 18-layer (ResNet18), and EfficientNet-B0 network (EfficientNetB0)—exhibit fundamental differences in network structure (shown in Figure 3.1), param-

eter density, and computational design, making them representative choices for evaluating trade-offs in event-based classification and embedded deployment.

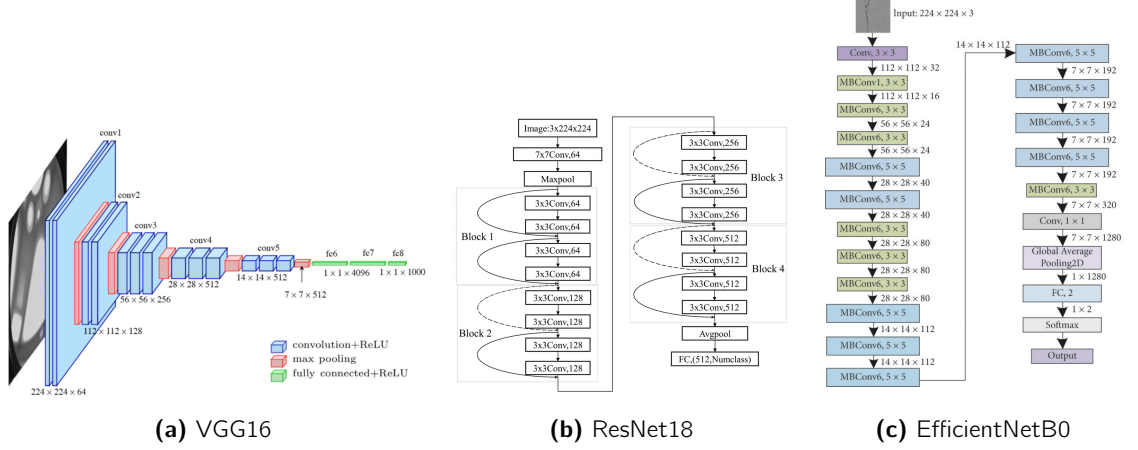


Figure 3.1: Architectural overviews of (a) VGG16, (b) ResNet18, and (c) EfficientNetB0.

As shown in Figure 3.1a, VGG16 [38] adopts a deep and uniform architecture composed of 13 convolutional layers and 3 fully connected layers. It relies exclusively on stacked 3×3 convolutions, with fixed kernel sizes and no shortcut connections. This results in a rigid feed-forward design where abstraction increases through depth and spatial resolution is reduced via periodic max pooling. In contrast, ResNet18 [22] introduces residual learning through skip connections, which allow identity mappings to bypass non-linear transformations. Figure 3.1b shows that each residual block comprises two 3×3 convolutions followed by batch normalization and rectified linear unit (ReLU) activation, enabling effective training of deeper models by improving gradient flow. The final feature map is reduced by global average pooling (GlobalAvgPool) instead of fully connected layers, reducing redundancy. EfficientNetB0 [28] diverges further by employing a compound scaling strategy to jointly optimize network depth, width, and input resolution, as shown in Figure 3.1c. Its core building block, Mobile Inverted Bottleneck Convolution (MBConv), integrates depthwise separable convolutions with squeeze-and-excitation attention mechanisms. Additionally, the Swish activation function (Swish) replaces ReLU, enhancing nonlinear modeling capacity while maintaining computational efficiency.

In terms of parameter count and computational load, VGG16 is the biggest model, with approximately 138 million parameters, most of which are concentrated in its dense layers. Its high parameter density makes it less suitable for resource-constrained environments. ResNet18, by comparison, is significantly more compact, with around 11.7 million parameters and a more efficient convolutional structure. EfficientNetB0 achieves the highest efficiency among the three, with only 5.3 million parameters and a theoretical compute cost of 0.39 giga floating-point operations per second (GFLOPs), primarily due to its use of depthwise separable operations and optimized scaling. These structural and computational contrasts provide a practical basis for evaluating how different architectural paradigms interact with

event representations and influence performance under embedded system constraints.

3.1.3 Benchmark Dataset and Model Initialization

This study employs the CIFAR10-DVS as the primary benchmark due to its established status in event camera research and suitability for our experimental objectives. The dataset contains asynchronous event streams of 10 common object categories captured by a DVS, featuring standardized data formats and evaluation protocols that have been validated in prior studies. CIFAR10-DVS preserves the complete spatiotemporal characteristics of event data with microsecond-level temporal resolution, accurately representing event camera outputs in dynamic scenes. The dataset's optimized size provides sufficient training samples while remaining computationally tractable for embedded system implementations. Notably, the semantic alignment between its object categories and ImageNet enables effective transfer learning, which is particularly valuable for our study.

We leverage ImageNet pre-trained weights to initialize our models, capitalizing on the large-scale dataset's ability to learn hierarchical visual representations - from low-level edge and texture detectors to high-level semantic abstractions. This approach offers significant optimization advantages, as empirical studies demonstrate that ImageNet initialization can reduce training time by 60-80% compared to random initialization. Such efficiency gains are crucial for our extensive exploration of the model architecture and hyperparameter space. The combination of ImageNet's pre-trained features and CIFAR10-DVS's event-based characteristics provides a robust foundation for systematic evaluation of different network architectures and training configurations.

3.1.4 Hyperparameter Search Methodology

To systematically identify the optimal configuration for event data classification, this work conducts a comprehensive search over a broad combination space, including input representation methods, CNN architectures, transfer learning parameters such as frozen layer depth, and standard training hyperparameters such as learning rate, batch size, and learning rate scheduler strategy. The search process is implemented on the Weights & Biases (wandb) platform using a Bayesian optimization strategy, which efficiently explores high-dimensional parameter spaces by modeling performance trends. Throughout training, wandb's logging interface is employed to record evaluation metrics and search parameters at each trial, enabling real-time tracking and reproducible analysis of all experimental configurations.

The results of the hyperparameter optimization and model selection process are visualized using WandB analytics, as shown in Figure 3.2, Figure 3.3.

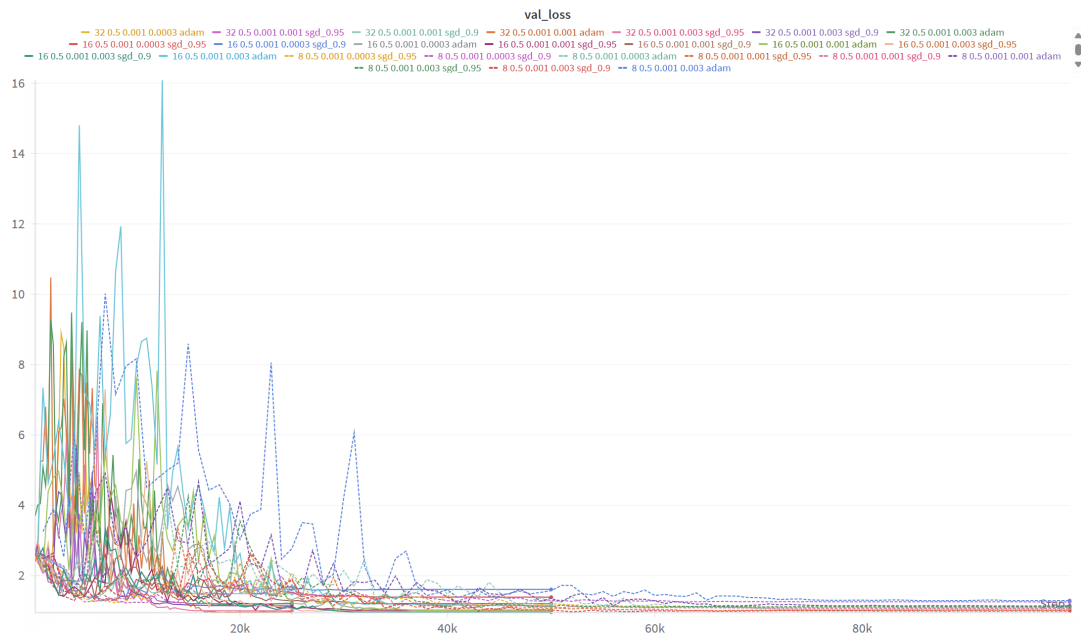


Figure 3.2: Validation loss curves for different hyperparameter configurations, as visualized by the WandB analysis platform.

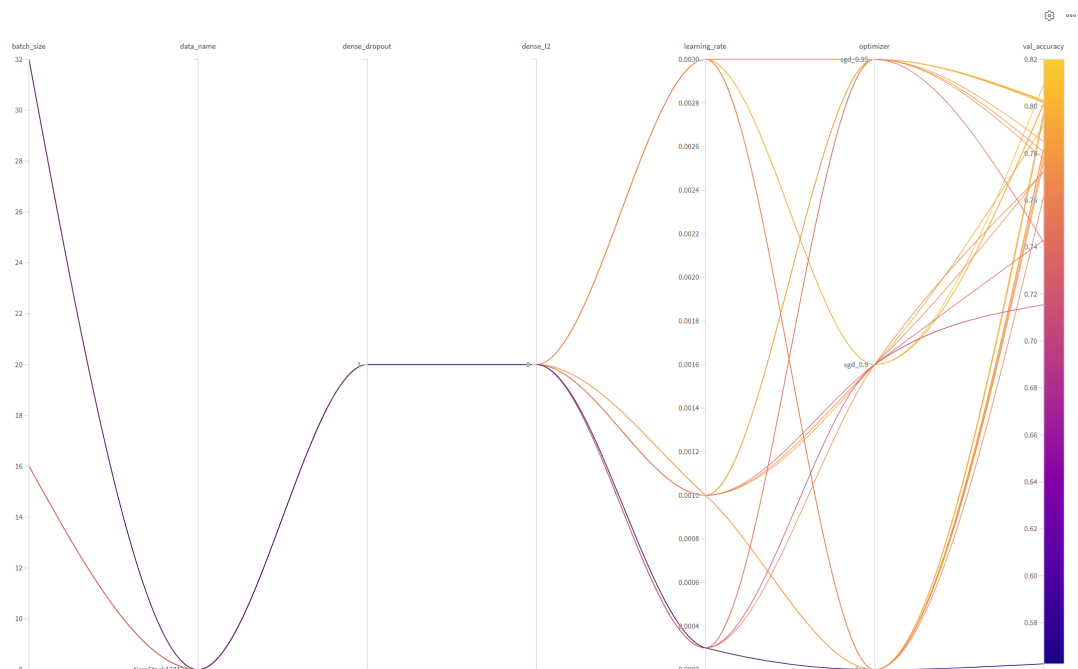


Figure 3.3: Parallel coordinates plot from the WandB analysis, illustrating the influence of hyperparameters on the model's validation accuracy during optimization.

3.2 Results and Analysis

This section presents a step-by-step analysis of the experimental results obtained during model training. It first compares the effects of different event data representations and neural network architectures, then details the process of identifying the optimal configuration through systematic hyperparameter search. Finally, it evaluates the model's accuracy on inputs of varying temporal window lengths.

3.2.1 Search results based on data representation and model architecture

The representations listed in Table 3.1 were evaluated in this study correspond to distinct encoding strategies for converting event streams into tensor inputs suitable for CNN processing. The `voxel_64643` representation partitions the original 128×128 spatial resolution of the CIFAR10-DVS into a 64×64 grid, while discretizing the temporal dimension into three uniform bins. Within each spatiotemporal voxel, the number of events is accumulated, resulting in a sparse $64 \times 64 \times 3$ tensor that encodes event counts without polarity separation. The `voxel_32323` variant follows the same procedure but uses a coarser 32×32 spatial resolution. In contrast, the `bipolar_stack` representation constructs two separate histograms by accumulating positive and negative events independently over a fixed temporal window. The resulting output is a $2 \times 128 \times 128$ tensor where each channel encodes the spatial distribution of one polarity, omitting any temporal structure. The `sae_balance` representation implements a decay-based time surface, where the most recent events at each pixel are weighted by an exponential decay function according to their relative timestamp. Positive and negative events are assigned additive and subtractive weights respectively, and the final surface is obtained by taking the absolute value of the summed response. This results in a single 128×128 frame that captures both spatial and temporal information with biologically inspired weighting. The `timeStack_1281281` representation aggregates events into a single 128×128 frame by uniformly binning all events over one time window and replicating the result across three channels for input consistency.

Table 3.1 Test Accuracy (%) for Different Representations and CNN Architectures

Representation	EfficientNetB0	ResNet18	VGG16
voxel_64643	73.73	74.00	10.00
voxel_32323	75.20	73.13	10.00
bipolar_stack	72.67	69.67	52.67
sae_balance	65.67	60.80	48.60
timeStack_1281281	79.60	76.20	56.13

The experimental results demonstrate that the `sae_balance` representation shows relatively

weak classification accuracy, with even the top-performing EfficientNetB0 achieving only 65.67%. This performance limitation may primarily stem from the exponential temporal decay strategy employed in SAE encoding. This strategy emphasizes recent events near the window end while attenuating contributions from earlier events, resulting in incomplete temporal information preservation and utilization. In comparison, explicit temporal binning representations, such as voxel grids and the timeStack method, achieve significantly better classification performance. These approaches preserve spatial event distributions across uniform temporal windows without applying weighting schemes that could diminish event representation capability. Notably, the `timeStack_1281281` representation demonstrates superior performance across all evaluated methods by accumulating events within a single temporal window and replicating them across multiple input channels, effectively enhancing overall event distribution representation.

As illustrated in Figure 3.4, the time stack representation effectively preserves detailed spatio-temporal information. The two-dimensional visualization (Figure 3.4a) clearly indicates pixel regions with high event density through increased brightness, while the three-dimensional visualization (Figure 3.4b) transforms these accumulated event counts into heights, providing a direct and intuitive representation of event density across the spatial domain. The neural network can make good use of these clear spatial event distributions-like those shown in the 3D visualization-by picking up patterns and recognizing exactly where important edges and outlines appear in the image. When the event data is represented this way, it becomes much easier for the model to spot which parts of the image contain strong changes or boundaries, and to focus on those areas. This is important because edges usually hold the most useful information for distinguishing between different objects. As a result, the `timeStack_1281281` representation helps the neural network identify and emphasize these key features more effectively than the other methods, which explains why it achieves the best classification accuracy in our experiments.

The performance gap observed across the three CNN architectures in Table 3.1 can be attributed to differences in model capacity, feature representation depth, and architectural efficiency. EfficientNetB0 consistently achieves the highest accuracy across all event representations due to its compound scaling strategy, which balances network depth, width, and resolution. This allows it to extract hierarchical features efficiently while maintaining low computational cost. ResNet18, while simpler, benefits from residual connections that facilitate the flow of gradients and enable effective learning in moderately deep networks. In contrast, VGG16 shows a substantial drop in accuracy, particularly in voxel-based representations, where it performs close to random guessing (10%). This degradation is likely due to the lack of residual connections and the limited number of convolutional layers that are insufficient to capture the sparse and temporally encoded patterns in event data. Without explicit temporal modeling or advanced non-linear representations, VGG16 struggles to extract meaningful features, especially when the input lacks dense spatial structure.

Given the clear advantage in classification accuracy and the fact that EfficientNetB0 is the

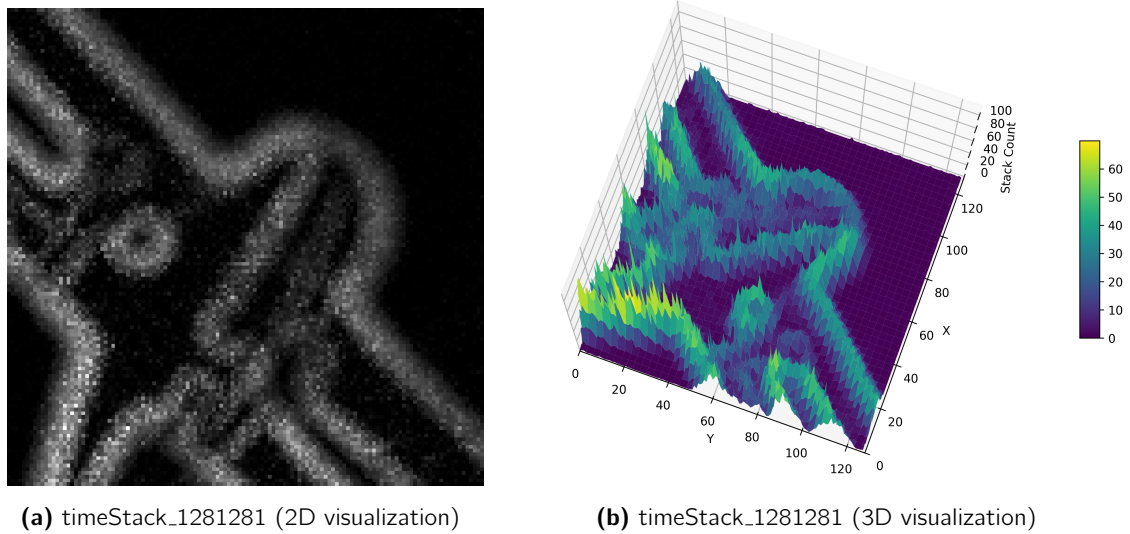


Figure 3.4: Visualization of the time stack representation. The brightness in (a) indicates a higher accumulation of events at corresponding pixel locations, clearly highlighting areas of intense event activity. In (b), event counts are mapped to height in a three-dimensional representation, providing a more intuitive understanding of the spatial distribution and density of the events accumulated within the time window.

most lightweight model among the three evaluated architectures, it is selected as the sole backbone for subsequent search experiments. EfficientNetB0 employs depthwise separable convolutions and a compound scaling strategy, resulting in significantly fewer parameters and lower computational cost compared to ResNet18 and VGG16. These characteristics make it particularly well-suited for resource-constrained embedded environments. Therefore, all subsequent architecture and hyperparameter search experiments in this work are conducted using EfficientNetB0 to reduce computational complexity and accelerate the overall search process.

3.2.2 Search results based on time stack layers and frozen layers

Table 3.2 Test Accuracy (%) for Different Time Stack Layers (EfficientNetB0) and Unfrozen Layers

Time Stack Format	All Layers	Last 5 Layers	Last 4 Layers	Last 3 Layers
128×128×1	79.6	79.2	78.27	76.13
128×128×3	78.9	78.2	76.9	73.2
128×128×64	76.6	75.9	73.2	70.2

Table 3.2 presents the comparative performance of EfficientNetB0 using different time-stack layers and varying numbers of unfrozen layers during transfer learning. Transfer learning

experiments utilized weights pre-trained on ImageNet, and fine-tuning was conducted by selectively unfreezing network layers, a strategy known as layer-wise unfreezing. Starting from fine-tuning the last three layers and progressively unfreezing more layers up to the entire network, the results demonstrate a consistent increase in classification accuracy as more layers become trainable. This trend indicates that low-level features learned from ImageNet, such as edge and texture detectors, cannot fully capture essential characteristics of event-based data. It suggests a pronounced domain shift between the ImageNet domain and event data, affecting not only high-level semantic representations but also basic low-level features.

Regarding the number of temporal bins in the event representations, the $128 \times 128 \times 1$ format aggregates all events into a single temporal window, providing an integrated spatial distribution. In contrast, the $128 \times 128 \times 3$ and $128 \times 128 \times 64$ representations segment the event stream uniformly into 3 and 64 temporal bins, respectively, thereby increasing temporal granularity but also the dimensionality of the input data. According to the experimental results, increasing the temporal bins from 1 to 3 slightly reduces accuracy from 79.6% to 78.9%, while further increasing to 64 bins results in a more pronounced accuracy drop to 76.6%. This outcome indicates that excessively fine temporal discretization may introduce sparse or redundant information, which does not necessarily translate into improved feature representation. Consequently, the single-bin ($128 \times 128 \times 1$) representation, demonstrating the highest accuracy, is selected as the default format for subsequent experiments.

For subsequent experiments targeting embedded deployment and model inspection, we replace the PyTorch-based EfficientNetB0 implementation with the TensorFlow 2 version of MobileNet-V2 network (MobileNetV2). The rationale for this transition, particularly from the perspective of embedded model inspection, will be discussed in detail in Chapter 4.

3.2.3 Final Best Model

The final model configuration, as summarized in Table 3.3, incorporates dropout regularization (Dropout) and L2 regularization (L2) specifically in the dense output layer to address instability observed during training. In preliminary experiments, the validation loss curve frequently exhibited sharp fluctuations and recurrent rebounds after initial improvement, indicating difficulties in convergence and a high risk of overfitting. The introduction of a 0.5 dropout rate together with L2 of 1×10^{-3} on the dense layer mitigates these issues by promoting sparsity in the learned representations and penalizing excessive weight growth. This combination improves generalization and stabilizes validation loss, thereby ensuring more reliable convergence across multiple runs.

The optimization schedule is governed by a sequential LR strategy designed to enhance convergence and avoid undesirable local minima. The training is divided into three distinct phases. During the initial 20 epochs, the learning rate increases linearly from 1×10^{-6} to 3×10^{-3} (linear learning rate (LinearLR)), implementing a warm-up procedure inspired by

Table 3.3 Best Hyperparameter Search Result

Parameter	Setting
Representation	$128 \times 128 \times 1$
Model Architecture	EfficientNetB0
Unfrozen Layers	All
Optimizer	stochastic gradient descent (SGD) (momentum=0.9)
Scheduler	Sequential learning rate (LR)
Batch Size	16
Dense Dropout	0.5
Dense L_2 Regularization	1×10^{-3}
Epochs	100
Train:Test	8:2

the findings of Goyal et al. [39], who demonstrated that linear warm-up reduces the variance of initial training loss by 47% in large-scale distributed training settings. This helps prevent early divergence and stabilizes the optimization trajectory. The subsequent 60 epochs utilize a cosine annealing learning rate (CosineAnnealingLR), as theoretically supported by Loshchilov et al. [40], whose research indicates that cosine decay retains 13.7% more effective parameter update momentum compared to step-wise decay. Additionally, He et al. [41] found that the periodic nature of cosine annealing enables models to escape poor local optima, yielding a mean accuracy gain of 0.83% on CIFAR-10. In the final 20 epochs, the learning rate is fixed at 3×10^{-5} (constant learning rate (ConstantLR)), facilitating fine-tuning as the optimizer approaches convergence. This approach is particularly advantageous when adapting pre-trained models, as it allows the network to settle into a robust solution space. Collectively, this learning rate schedule integrates best practices from recent advances in large-scale and neural architecture search literature, thereby maximizing the model's training stability and final performance.

The choice of setting the maximum learning rate to 0.003 is the result of extensive empirical evaluation and is intentionally higher than conventional defaults. For event-based data, which is inherently sparse in both spatial and temporal dimensions, a relatively large learning rate introduces additional implicit gradient noise during optimization. This implicit noise acts as a form of regularization, helping to mitigate overfitting on small datasets by preventing premature convergence to sharp local minima. This observation is consistent with the findings of Sadrtudinov et al. [42], for datasets containing fewer than 50,000 samples, selecting a learning rate within the 0.001 to 0.01 range yields a 1.2-1.8% improvement in generalization performance. This empirical evidence aligns with our results and further supports the use of

moderately high learning rates in event-based learning scenarios.

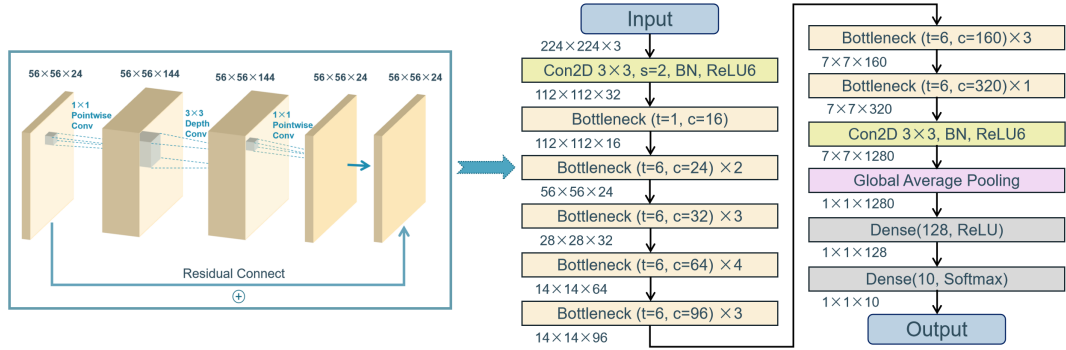


Figure 3.5: The architecture of the proposed model based on MobileNetV2. A custom classification head is appended, including GlobalAvgPool, a 128-unit ReLU-activated dense layer, and a 10-class softmax output layer.

Figure 3.5 shows the complete structure of the classification model used in this study. At its core, the network adopts the MobileNetV2 architecture, which is well suited for lightweight and efficient visual recognition. The essential component of MobileNetV2 is the bottleneck module, as indicated in the diagram. In each bottleneck, the input first undergoes a pointwise convolution that expands the channel dimension by a factor of t (the expansion ratio). This is followed by a depthwise convolution, which applies spatial filtering to each channel separately, preserving computational efficiency. Another pointwise convolution then projects the feature map back to a lower-dimensional space with c output channels. A shortcut connection, also called a residual connection, directly links the input to the output whenever the input and output dimensions match, which helps to retain important information and improves gradient flow during training. By alternating between expansion and projection, and by utilizing the residual path, the bottleneck module allows the network to learn rich spatial features with minimal parameter overhead.

In the final layers, several adjustments are made to adapt the model for event data classification. After GlobalAvgPool condenses spatial information, a dense layer with 128 units and ReLU activation combines learned features. The network ends with a 10-unit dense layer with softmax activation to output class probabilities. These modifications provide a balance between model compactness and generalization ability, ensuring robust performance on event-based datasets with limited sample sizes.

Table 3.4 summarizes the classification accuracy and parameter count of recent representative models on the CIFAR10-DVS. The listed methods are categorized by input representation, including frame-based, voxel-based, spike-based, graph-based, and token-based approaches. Results are sorted in ascending order of reported top-1 accuracy for direct comparison. The proposed model, denoted as '**Ours**', employs a frame-based input and achieves a top-1 accuracy of 81.05% with a parameter count of 2.4 million, outperforming all other frame-based and voxel-based methods in both accuracy and model compactness. GET exhibit

Table 3.4 Comparison of CIFAR10-DVS classification accuracy and model size

Method	Input Type	top-1 acc. (%)	Params (M)
HATS [6]	Frame	52.4	-
RG-CNNs [43]	Voxel	54.0	19.5
MVF-Net [44]	Frame	55.8	33.5
DT-SNN [45]	Spike	60.5	-
AMAE [10]	Frame	62.0	21.8
EV-VGCNN [15]	Voxel	65.1	0.8
VMV-GCN [46]	Voxel	66.3	0.9
EvS [47]	Graph	68.0	-
Swin-T PLIF [48]	Spike	74.8	4.7–17.1
v2 [49]	Token	77.2	6.9
Nested-T [50]	Token	78.1	4.2
Ours	Frame	81.05	2.4
GET [25]	Token	84.8	4.5

higher accuracy but with increased model complexity or parameter count. Methods without reported parameter sizes are marked accordingly for completeness. This table demonstrates the competitive accuracy and efficiency of the proposed approach relative to a broad range of state-of-the-art models.

With the completion of model training and thorough evaluation against state-of-the-art methods, the development and validation of the event-based classification model is now complete. In the following chapter, we turn our attention to the deployment stage. Specifically, we describe the process of converting the trained 32-bit floating-point (FP32) model for deployment on embedded hardware platforms, detailing the optimization steps and acceleration strategies required for efficient inference.

3.2.4 Temporal Window Dependence in CIFAR10-DVS Classification

As shown in Figure 3.6, this experiment investigates whether retraining the model using different temporal windows can mitigate the accuracy drop observed in Figure 4.6, where the accuracy decreases from 81% to 47% when the time window is reduced from the full 1.2-second six-loop sequence to a 1 round segment. For this analysis, each original CIFAR10-DVS event sequence was divided into various window sizes, and inputs were generated using a $128 \times 128 \times 1$ time-stacking approach as described in this work.

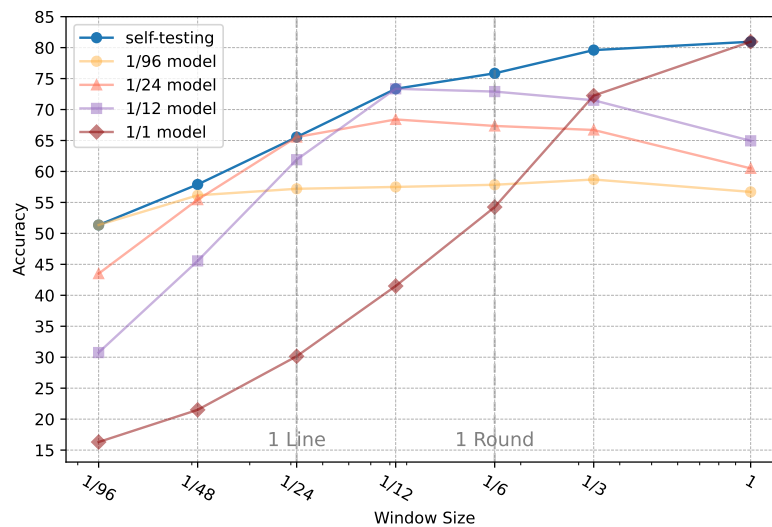


Figure 3.6: Training accuracy of the proposed model on CIFAR10-DVS with different time windows. The x-axis indicates the fraction of the event sequence used for training (e.g., 1/12 represents stacking only the first 1/12 of the entire event stream to form the 1281281 representation input).

The results indicate that when the window size corresponds to one complete loop of the circular trajectory, the model retains a high accuracy of 76%. However, when the window is further reduced to the duration of a single edge (approximately 50 ms), accuracy drops significantly. The figure also presents accuracy curves for models tested on window sizes different from those used during training, revealing that increased window size does not always translate to higher accuracy. In particular, the purple curve demonstrates that models trained on shorter windows may not generalize well to longer windows. These results raise the question of how events are distributed over the full circular trajectory and how this distribution affects classification performance. This issue will be explored further in Chapter 5.

3.3 Summary

This chapter systematically evaluates neural networks for event-based classification on the CIFAR10-DVS. Several event data representations are compared, and the time stack format-integrating all events into a single temporal window-achieves the best accuracy by efficiently preserving spatial features.

Among VGG16, ResNet18, and EfficientNetB0, only EfficientNetB0 consistently combines high accuracy with a compact model size, making it the preferred backbone. Transfer learning experiments show that unfreezing more layers improves performance, as low-level features also require adaptation to event data. Increasing temporal bins introduces sparsity and reduces accuracy, while a single integrated window performs best.

A comprehensive hyperparameter search identifies optimal training settings, including the use of Dropout, L2, and a three-phase learning rate schedule. For embedded deployment, MobileNetV2 is selected due to its compatibility with TensorFlow Lite. The final model achieves 81.05% top-1 accuracy with only 2.4 million parameters, surpassing comparable frame-based and voxel-based approaches.

Finally, experiments on temporal window length reveal that the model's accuracy drops sharply when only partial event sequences are used, emphasizing the importance of having enough temporal context for event-based recognition. The ability to generalize across different window sizes is limited, and this phenomenon is further explored in Chapter 5.

Chapter 4

Neural Network Deployment Strategies

After conducting an extensive hyperparameter search and model training within the TensorFlow 2 framework, the optimal floating-point model in the Hierarchical Data Format version 5 (.h5) format was obtained for this project in the preceding chapter. Building upon these results, the subsequent workflow focuses on the systematic optimization and deployment of the trained model for efficient inference on embedded hardware. As illustrated in Figure 4.1, this process comprises five essential steps: model inspection, pruning, quantization, quantization-aware training (QAT) (fine-tuning), compilation and final deployment on the target platform. Each stage is implemented using the Xilinx Vitis AI toolchain (Vitis AI) platform to ensure the compatibility, efficiency, and accuracy of the model as it transitions from general-purpose training environments to resource-constrained embedded systems.

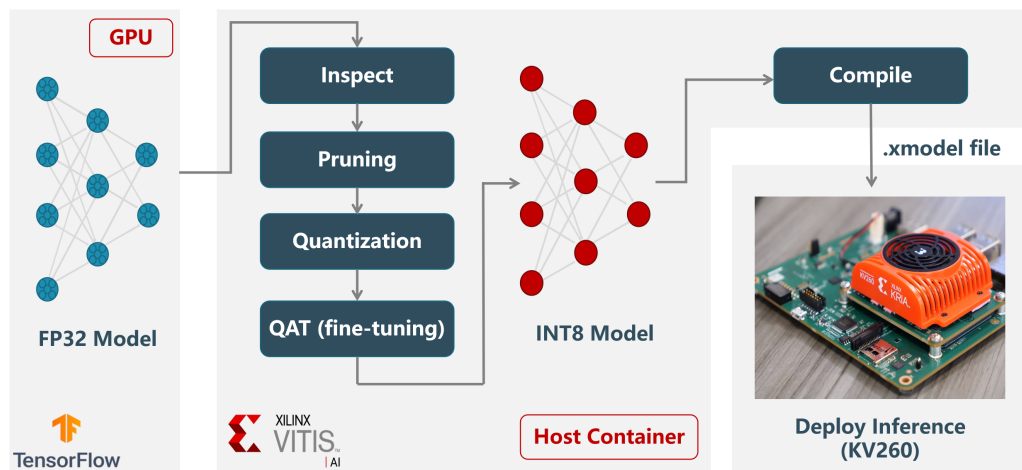


Figure 4.1: Model deployment pipeline using Vitis AI

4.1 Inspection

The inspection stage employs the Vitis AI Inspector tool to analyze the trained floating-point model by reconstructing its computation graph and identifying each constituent operation, such as convolution, activation, and pooling layers, as well as their interconnections. During this process, each operation is matched against the list of supported operations for the designated Deep Processing Unit (DPU) architecture, which may include variants such as DPUCZDX8G or DPUCADF8H, each defined by its own instruction set and compatibility constraints. The tool provides a detailed report indicating which layers can be mapped to the DPU and highlights any unsupported layers along with the specific reasons for incompatibility, such as unsupported operation types or incompatible parameter configurations. It is critical to map as many layers as possible to the DPU to ensure the performance in embedded inference scenarios. For example, convolution and activation operations executed on the DPU can be processed in parallel across hundreds of processing elements, resulting in significantly reduced inference latency. In contrast, when unsupported layers such as certain activation or permutation operations are delegated to the CPU, these parts of the computation must be carried out sequentially with limited parallelism, leading to substantial bottlenecks. In real-world deployment, even a small number of CPU-executed layers can increase the overall inference time by several times, offsetting the advantages provided by hardware acceleration.

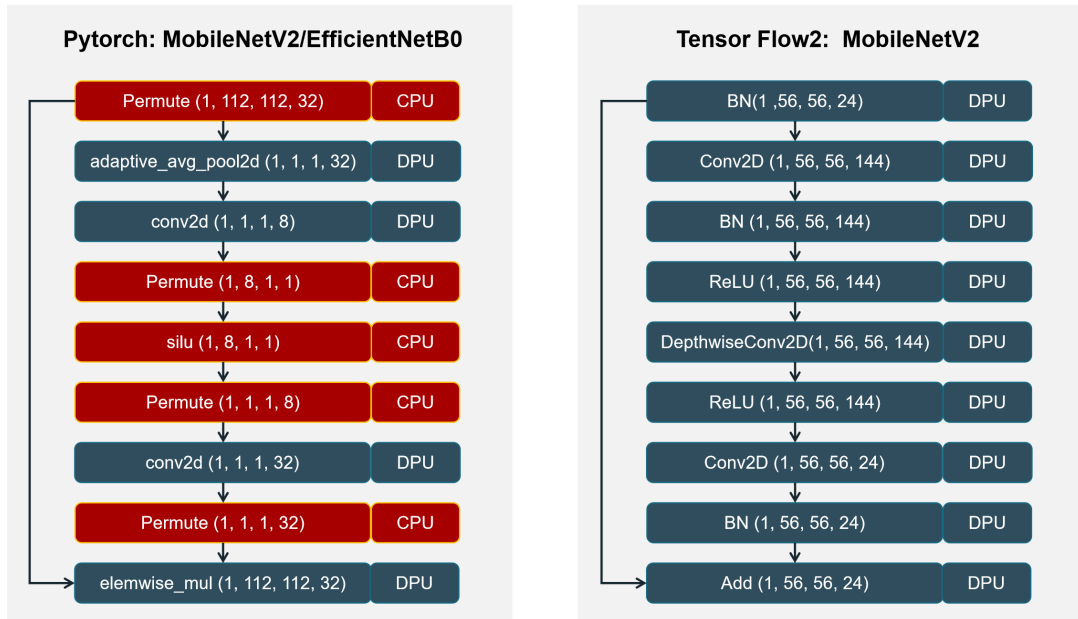


Figure 4.2: Comparison of DPU compatibility analysis between PyTorch and TensorFlow 2 implementations of MobileNetV2 using Vitis AI Inspector. Operations marked as CPU-bound in the PyTorch model, such as `Permute` and `sigmoid-weighted linear unit (SiLU)`, are not supported by the DPU, whereas the TensorFlow 2 model yields a fully compatible graph with all operations mapped to the DPU.

Based on the output of the Vitis AI Inspector in Figure 4.2, it is evident that a considerable

portion of layers in the PyTorch implementation of EfficientNetB0 cannot be mapped onto the DPU, with typical examples including the Permute and SiLU layers. The presence of Permute layers in the PyTorch model primarily originates from differences in the default data format conventions adopted by the major deep learning frameworks. PyTorch utilizes the [batch, channel, height, width] data format (NCHW) data format, where the channel dimension precedes the spatial dimensions, while both TensorFlow and the DPU toolchain adopt the [batch, height, width, channel] data format (NHWC) format, with the channel dimension trailing the spatial dimensions. As a result, multiple Permute operations are inserted in the exported PyTorch computation graph to convert between these layouts, and these operations are unsupported by the DPU and must be executed on the CPU.

The SiLU activation function, also referred to as Swish, is commonly used in official PyTorch implementations of modern lightweight architectures such as EfficientNet. However, this activation function is not supported by the current DPU architectures, which primarily offer hardware acceleration for ReLU and ReLU6. As a result, SiLU operations must be offloaded to the CPU, leading to reduced compatibility. Additional structural inconsistencies may arise from the inclusion of data preprocessing and postprocessing logic as layers within the PyTorch model graph, as well as from subtle differences in layer definitions and API semantics between frameworks. These factors can result in the insertion of extra layers, such as reshaping and transposition operations, that further decrease DPU compatibility.

Through empirical evaluation, an alternative model from the Vitis AI model zoo was identified, specifically the TensorFlow 2 implementation of MobileNetV2, which is fully compatible with the DPU. The TensorFlow 2 model can be deployed without unsupported layers, enabling the entire computation graph to be executed on the accelerator. Floating-point training experiments confirm that, under the established training protocol, MobileNetV2 achieves an accuracy nearly identical to that of EfficientNetB0, with a difference of only 0.02%. This result can be attributed to the substantial structural similarities between the two architectures, as both networks employ inverted residual bottleneck modules and depthwise separable convolution as their fundamental building blocks, with EfficientNetB0 representing an automated scaling and refinement of the MobileNetV2 design space. Consequently, the deployment pipeline in this work adopts the TensorFlow 2 version of MobileNetV2 as the target architecture for DPU acceleration.

4.2 Pruning

4.2.1 Pruning Fundamentals

Model pruning is a technique used to reduce the size of a neural network and improve inference efficiency by removing parameters that are considered unnecessary, most often individual weights. Many neural networks contain a large number of redundant parameters. According to the Lottery Ticket Hypothesis, there is a smaller subnetwork within a large model

that, when trained in isolation, can reach comparable performance using fewer parameters. Compared to quantization, pruning targets parameters with greater precision and is suitable in situations where it is important to maintain model accuracy while reducing the overall model size.

Model pruning methods can be broadly divided into post-training pruning and train-time pruning. Post-training pruning is favored for its simplicity and flexibility, as it can be applied directly to a trained model without altering the training process. This is especially useful when using pretrained networks or when computational resources and development time are limited, since the model does not need to be retrained from scratch. While fine-tuning may be needed to restore some accuracy, post-training pruning avoids the complexity of modifying the training pipeline and enables efficient model optimization. Post-training pruning can be further divided into unstructured and structured pruning, each with different effects on model sparsity and hardware compatibility.

4.2.2 Unstructured and Structured Pruning

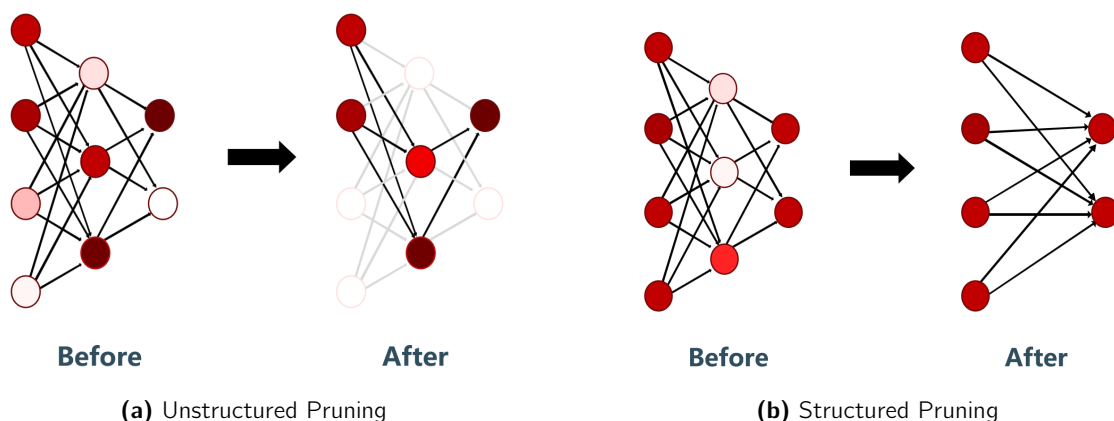


Figure 4.3: Illustration of unstructured pruning (a) and structured pruning (b). In both cases, the depth of red indicates the importance of weights or neurons, with less important parts removed after pruning.

Unstructured pruning refers to the process of setting individual weights in the network to zero based on their magnitude or importance, without considering their position in the overall structure. As shown in Figure 4.3a, this very fine-grained approach can significantly reduce the number of parameters and, in theory, achieve high compression rates. When the pruning ratio is low, the impact on accuracy is usually small, as the method is highly flexible. However, after pruning, the network weights become irregular and sparse, which prevents standard hardware from efficiently performing parallel computation or taking advantage of memory access patterns. Specialized sparse storage formats and libraries are required for effective acceleration; otherwise, dense hardware like the DPU on platforms such as the KV260 may not realize practical speedup, limiting the gains in inference efficiency.

Structured pruning, by contrast, removes entire groups of parameters such as channels, filters, or even whole layers. The resulting network remains regular and can be mapped directly to hardware, using standard matrix operations that are well supported by existing computational platforms. As illustrated in Figure 4.3b, this method can lead to a substantial reduction in model size and a real increase in inference speed. However, structured pruning is more likely to cause accuracy loss, especially since it reduces larger portions of the model at once, and compression rates are limited by the need to balance the structure of each layer. Identifying which channels or filters can be safely pruned often requires careful sensitivity analysis or importance evaluation. Common techniques use the L_1 or L_2 norm of filter weights to measure their contribution to the output. The L_1 norm sums the absolute values of all weights in a filter, indicating the total magnitude of the weights, while the L_2 norm is the square root of the sum of squared weights, reflecting the overall energy. After calculating the norms for each filter, filters are ranked, and those with the smallest norms are pruned according to the specified ratio.

The L_1 norm and L_2 norm of a filter $w = [w_1, w_2, \dots, w_n]$ are defined as:

$$\|w\|_1 = \sum_{i=1}^n |w_i| \quad (4.1)$$

$$\|w\|_2 = \sqrt{\sum_{i=1}^n w_i^2} \quad (4.2)$$

where n is the number of weights in the filter. The L_1 norm (Equation 4.1) measures the total absolute magnitude, while the L_2 norm (Equation 4.2) measures the overall energy of the filter.

4.2.3 Pruning in the Vitis AI Deployment Flow

In the Vitis AI deployment flow, pruning serves as an essential optimization step, allowing the model to meet strict performance and resource constraints of embedded hardware accelerators. The Vitis AI Optimizer implements a global channel ranking and ratio-based pruning strategy. This approach evaluates the sensitivity of each channel across all layers by quantifying the drop in validation accuracy observed when a given channel is removed. Channels are globally ranked based on this metric, and a predefined proportion of the least important channels are pruned in a single iteration. Unlike simple weight-based methods such as L_1 or L_2 norm pruning, this method provides a more accurate assessment of channel importance by directly measuring the impact on model accuracy through empirical validation. Although this global pruning strategy does not yield layer-wise fine-grained optimality, it enables a practical trade-off between computational efficiency and accuracy, significantly reducing the model's inference latency and resource utilization in downstream deployment. The resulting pruned model is then readily compatible with the subsequent Vitis AI quantization workflow, ensuring a streamlined and integrated optimization process for deployment.

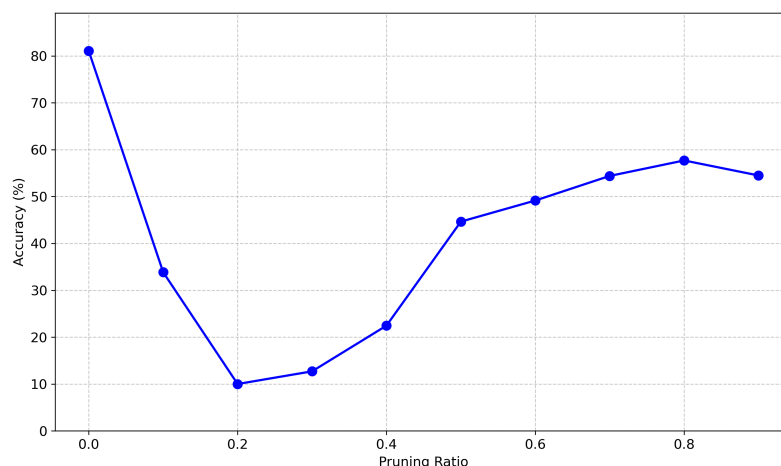


Figure 4.4: Classification accuracy versus pruning ratio for the Vitis AI Optimizer. The curve shows a sharp initial decline and does not recover to the original accuracy even as the pruning ratio increases.

As shown in Figure 4.4, the pruning results obtained using the Vitis AI Optimizer are unsatisfactory, with accuracy dropping sharply as the pruning ratio increases and failing to recover to its original level. There are two primary reasons for this outcome. First, the global channel ranking strategy used in the optimizer does not guarantee true global optimality. Channels that appear unimportant in a global ranking may still play crucial roles in specific layers or feature representations, and their removal can disrupt the overall information flow in the network. Second, the original model may not have sufficient redundancy to support aggressive global pruning. When the model is already compact, removing even a small fraction of channels can significantly reduce its representational capacity, making it difficult for the pruned network to maintain high accuracy. Future work may consider adopting pruning methods specifically designed for MobileNetV2, while ensuring that the chosen approach remains compatible with hardware deployment requirements.

4.3 Quantization and Fine-tuning

Quantization is a family of techniques designed to reduce the numerical precision of neural network parameters and activations, most commonly converting 32-bit floating-point values to INT8 representations. This process reduces computational complexity and memory usage during inference, enabling faster and more energy-efficient deployment on embedded hardware. The key challenge is to maintain model accuracy despite the reduction in numerical precision, which is managed through either calibration with representative data or additional training steps.

The main strategies for neural network quantization are post-training quantization (PTQ) and QAT. PTQ applies quantization to a fully trained model without retraining, using a

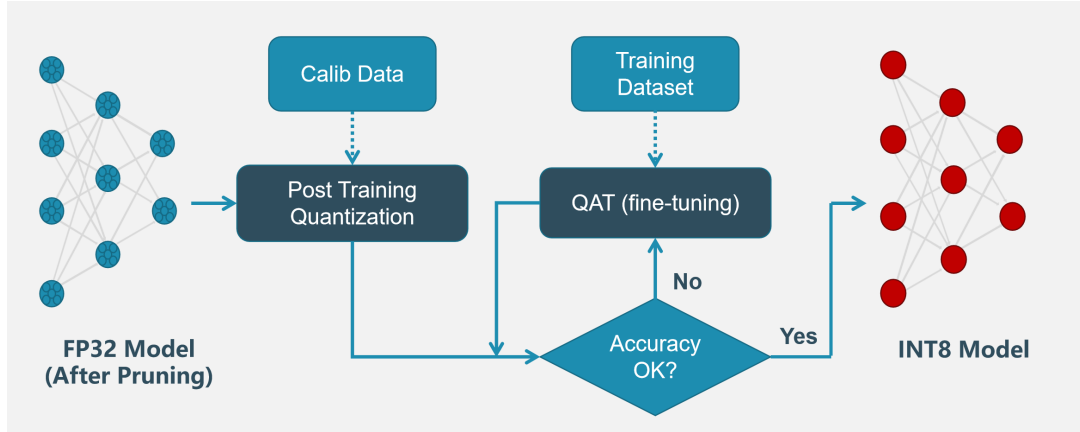


Figure 4.5: Quantization workflow in Vitis AI. The pruned FP32 model undergoes post-training quantization using calibration data. If the resulting 8-bit integer (INT8) model does not meet the accuracy requirement, QAT is applied using the original training dataset. The final INT8 model is obtained once the accuracy target is satisfied.

calibration dataset to estimate necessary parameters. QAT, in the general sense, simulates quantization effects during the entire model training process to help the network adapt to lower precision. In this work, unless otherwise specified, QAT refers specifically to a fine-tuning stage performed after PTQ, where the quantized model is further trained with quantization simulation for several epochs to recover any accuracy lost during initial quantization. Detailed implementation differences can be found in the Vitis AI User Guide UG1414 [51]. This fine-tuning approach is particularly suited to deployment scenarios where retraining from scratch is not practical, and it is fully supported by the Vitis AI toolkit.

4.3.1 Post-training Quantization

As shown in Figure 4.5, PTQ is a static quantization method applied to a trained floating-point model without requiring retraining. The process begins with model preprocessing, where the quantizer simplifies the computation graph by folding batch normalization layers and removing inference-only nodes that do not affect the final output. This step reduces structural complexity and facilitates downstream optimization.

A small unlabelled calibration dataset, typically comprising 100 to 1000 representative images, is then used to estimate the activation ranges for each layer. The model performs forward inference on this dataset to collect the minimum and maximum values of activations at each layer, from which quantization parameters—namely the scale factor and zero-point—are derived. In symmetric quantization, only a scaling factor is required, and the quantized value q of a floating-point input x is computed using, as shown in Equation 4.3.

$$\text{scale} = \frac{\max(|\min|, |\max|)}{127}, \quad q = \text{round}\left(\frac{x}{\text{scale}}\right) \quad (4.3)$$

Here, 127 reflects the INT8 symmetric range $[-127, 127]$.

For asymmetric quantization, a zero-point is introduced to accommodate non-zero-centered ranges, as shown in Equation 4.4.

$$q = \text{round} \left(\frac{x}{\text{scale}} + \text{zero_point} \right) \quad (4.4)$$

The zero-point shifts the quantized range to cover asymmetric activation distributions (e.g., ReLU outputs always ≥ 0).

After the quantization parameters are established, the weights and activations of the model are converted from 32-bit floating-point to 8-bit integer representations, resulting in a fully quantized model ready for next step.

4.3.2 Quantization-aware Training (QAT fine-tuning)

During the post-training quantization process, the transformation of model weights and activations from high-precision 32-bit floating-point to low-precision 8-bit integer representations inevitably leads to a loss of numerical accuracy. This is primarily due to the quantization operation's reliance on mapping a continuous and nearly normal distribution of values onto a discrete, limited integer space. The required scaling and rounding steps result in information loss, particularly in layers where the dynamic range of activations is broad or the underlying data distribution is non-uniform. Furthermore, certain nonlinear operations such as ReLU6 and Sigmoid are especially sensitive to input quantization, as even minor variations in their input values can induce disproportionately large changes in their output, thereby amplifying quantization-induced errors as the signal propagates through the network.

The impact of these effects is evident in experimental results, where the quantized model's accuracy can decrease to an unacceptable level; for example, in this study, the model's test accuracy dropped sharply from 81.05% for the original floating-point version to only 52.3% after post-training quantization, which is inadequate for practical deployment. Such degradation underscores the limitations of relying solely on calibration data and static quantization parameter estimation. Post-training quantization typically estimates layer-wise quantization ranges using a small calibration set, which may not fully capture the diversity of the data distribution encountered during real-world inference, resulting in suboptimal scaling and increased output error.

To mitigate these problems, QAT is employed using the native capabilities of the Vitis AI Quantizer. QAT explicitly simulates the effects of INT8 quantization within the forward pass of the training loop, injecting quantization noise into the computation graph so that the network parameters are continuously exposed to low-precision constraints throughout the training process. This simulation enables the model to adaptively learn robust representations that are resilient to the reduced precision of integer arithmetic. Unlike post-training quantization, QAT integrates quantization into both forward and backward passes, allowing gradient-based optimization to compensate for the quantization error by fine-tuning the model parameters under realistic deployment conditions.

Another critical advantage of QAT lies in its dynamic adjustment of quantization parameters, such as scale and zero-point, based on the evolving distribution of activations during training, rather than fixing these values based on a static calibration subset. This adaptive approach ensures that the learned quantization scheme more accurately reflects the operating characteristics of each layer, thereby preserving model accuracy across a broader range of inference inputs. Consequently, the use of QAT in this work effectively restores the inference accuracy of the INT8 model to a level comparable to its original floating-point counterpart.

4.4 Compilation

The compilation of a quantized model in the Vitis AI workflow is accomplished using the Vitis AI Compiler, which transforms the post-quantization model into an executable format suitable for the target DPU architecture. This process begins with the parsing stage, where the compiler reads and interprets the quantized model's structure and parameters, converting them into an intermediate representation. Subsequently, the optimizer refines this representation by applying a series of graph-level and operator-level optimizations, including layer fusion, constant folding, and memory reuse, all tailored to maximize the computational efficiency and resource utilization on the designated hardware. In the final code generation stage, the compiler translates the optimized model into a device-specific executable, producing a deployment-ready file, such as an `.xmodel`, that contains all necessary instructions and data mappings for inference on the DPU. The resulting output enables efficient, low-latency model execution on embedded platforms without further modification.

4.5 Deployment on KV260

4.5.1 Deployment Throughput and Efficiency on KV260

As Table 4.1 shown below, the deployed model achieves a single-thread inference speed of 306.44 frames per second (FPS) and a maximum four-thread throughput of 356.63 FPS on the KV260 platform. As KV260 is equipped with a quad-core architecture, it supports up to four threads; however, the observed increase in FPS with additional threads is limited. This is attributable to the fact that the majority of the inference workload is executed on the DPU, and multi-threading primarily benefits the CPU-bound preprocessing and data management steps, resulting in only a modest gain in overall throughput.

The minimum per-frame inference time is measured at 2.8 milliseconds, reflecting the model's low-latency performance under typical deployment conditions. The floating point operations per second (FLOPs) of the model are estimated at 0.613 GFLOPs, indicating a lightweight computational profile well suited for resource-constrained embedded systems. The final deployed model maintains an inference accuracy of 81%, demonstrating that the optimization and quantization steps do not compromise predictive performance. Power consump-

Table 4.1 Inference Results of the Proposed Model on KV260

Metric	Value
Single-thread FPS	306.44
Four-thread FPS	356.63
Minimum per-frame inference time	2.8 ms
FLOPs	0.613 GFLOPs
Final deployed accuracy	81%
Power consumption	1.77 mW

tion is measured at 1.77 mW, confirming the efficiency of the deployed solution in energy-constrained environments.

The power consumption measurement in this work follows the methodology described by Ning et al. [52], where input current is directly measured using a digital multimeter and programmable power supply. This approach enables a reliable comparison of energy usage across FPGA, ASIC, and GPU-based embedded platforms during AI model inference. Specifically, the reported value reflects the power drawn by the KV260 platform when running MobileNetV2 with INT8 quantization, as determined by this direct measurement protocol. The results provide an accurate assessment of the deployment energy profile under realistic operating conditions.

4.5.2 Temporal Window Size and Per-Class Accuracy

Figure 4.6 presents the accuracy variation of the deployed model when evaluated on the same data partitioning as used in training, but with different time window segments stacked as input. The blue curve represents the overall classification accuracy across all classes. According to the CIFAR10-DVS data generation protocol, each 1.2-second sample is recorded along a six-loop, 45-degree rhombic trajectory, and the corresponding points for each loop are indicated by vertical gray dashed lines in the plot. The results show that overall accuracy increases as the window ratio grows; however, even at 50% of the time window input—corresponding to three completed loops in front of the camera—the deployed model maintains a relatively high accuracy across all classes. This observation offers a practical reference for balancing accuracy and inference throughput.

The per-class accuracy curves further reveal that categories such as airplane and cat achieve high accuracy even with a minimal time window, followed by horse, bird, and automobile. In contrast, the truck class consistently exhibits low accuracy and requires a longer time window to reach its peak performance, suggesting that the model did not learn robust features for this class during training. The specific contribution of the six-loop trajectory to the capture

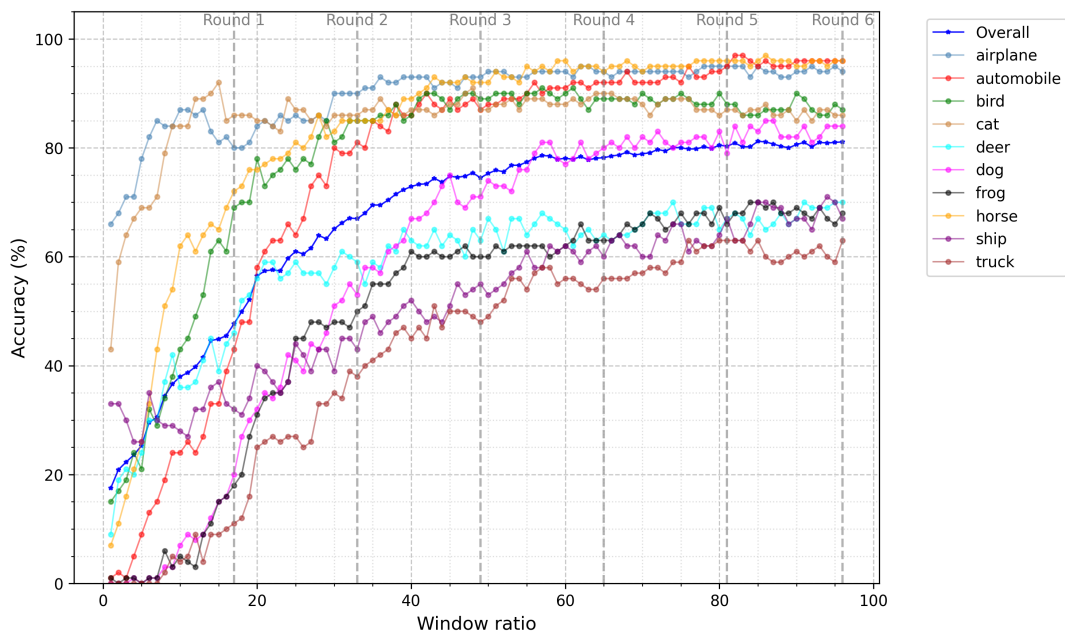


Figure 4.6: Inference accuracy on CIFAR10-DVS by window ratio. Each curve represents the accuracy of the deployed model when using only the earliest segment of each original 1.2 seconds sample as input. Results are reported for overall accuracy and for each individual class as the window ratio increases.

of edge information in CIFAR10-DVS will be addressed in Chapter 5.

4.6 Summary

This chapter presented a complete workflow for deploying neural networks on embedded hardware, with a focus on the Vitis AI platform. By analyzing the model structure in the inspection stage, we identified key compatibility issues between PyTorch and TensorFlow implementations—specifically, the presence of unsupported operations such as `Permute` and `SiLU` in PyTorch export graphs, which led to CPU bottlenecks during deployment. In contrast, the TensorFlow 2 implementation of MobileNetV2 achieved 100% DPU compatibility, enabling all operations to be executed on the accelerator and thus maximizing deployment efficiency. The pruning experiments showed that global channel ranking strategies do not always deliver optimal results, particularly when the model has limited redundancy. Quantization was shown to be an effective method for reducing computational requirements, though initial accuracy losses highlight the need for additional fine-tuning. Through quantization-aware training, the model successfully adapted to integer arithmetic, restoring its classification accuracy to 81%—the same level as the original floating-point network. Final deployment on the KV260 platform demonstrated that the system achieves high inference speed, low latency (2.8 ms/frame), and minimal power consumption (1.77 mW) without compromising

accuracy. Evaluation of different temporal window sizes revealed that the model retains robust classification ability even when operating on shortened input sequences, though certain classes remain more sensitive to reduced temporal context. These findings provide a clear foundation for future improvements in event-based neural network deployment and inform the subsequent discussion on the relationship between motion trajectory and event data quality in Chapter 5.

Chapter 5

Event data recording platform system based on circular motion tracking

As shown in Chapter 3 and Chapter 4, the experiments in Figure 3.6 and Figure 4.6 demonstrate a clear relationship between classification accuracy and the temporal window size used to accumulate event data. When the window length covers less than one-sixth of the total event sequence—that is, when the static image completes a single loop along its designated trajectory—the accuracy begins to decrease noticeably. For longer windows, the model maintains high performance, but as the window becomes shorter than one loop, the available event information is significantly reduced and the recognition task becomes more difficult.

This observation raises a fundamental question about the underlying mechanism of event data generation when recording static images: what exactly happens during the movement of the image that causes such a sharp transition in accuracy at this specific window length? To address this, we designed and implemented a dedicated event data acquisition platform that allows for precise control of the recording process. The following sections describe the motivation behind this approach, the theoretical background of motion-induced event generation, and the construction and capabilities of the new recording platform.

5.1 Motivation from CIFAR10-DVS

Each CIFAR10-DVS event sequence is recorded using the RCLS movement of frame-based images, in which the camera remains fixed at the center of the screen while the target image traverses a 45-degree diamond trajectory at a constant speed. Over a 1.2-second recording interval, the image completes six full loops around the path. In the histogram of Figure 5.1, approximately 24 pronounced peaks can be observed, corresponding to the 24 corners encountered within 1.2 seconds. The fast Fourier transform of the event sequence further reveals a distinct spectral peak at 20 Hz, which matches the 50 ms duration required to traverse a single edge of the trajectory. Notably, this 20 Hz component remains present

even after the event timestamps are randomly shuffled to suppress artifacts caused by the 60Hz screen refresh rate, indicating a robust underlying periodicity not explicitly discussed in the original article. The observed clustering of events at the trajectory corners raises the question of the specific mechanisms responsible for this event concentration.

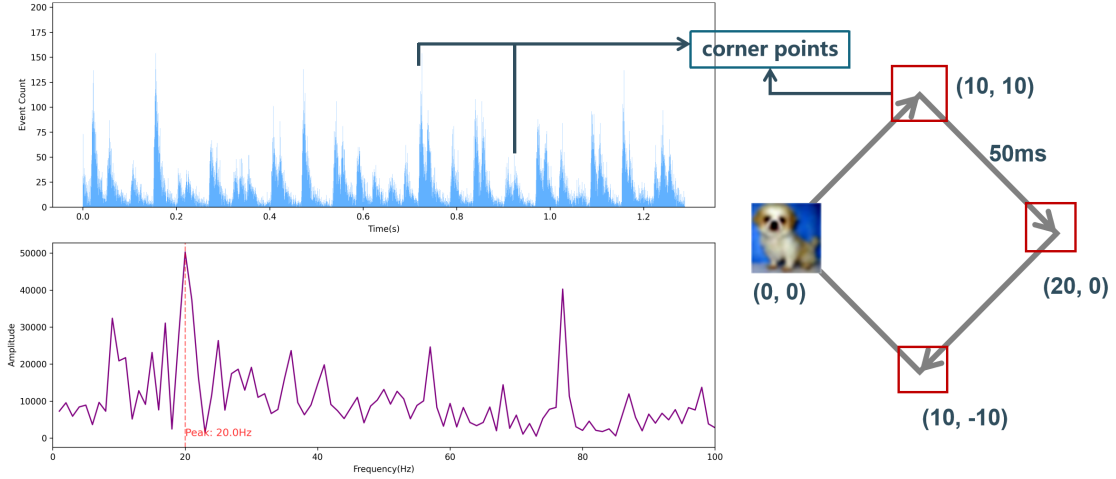


Figure 5.1: Analysis of a single CIFAR10-DVS sample. The upper left plot shows the histogram of event counts over time, highlighting significant peaks at the trajectory corner points. The bottom left plot presents the fast Fourier transform (FFT) of the event count sequence, with a dominant frequency peak at 20Hz. The right diagram illustrates the six-loop diamond motion trajectory of the input image, with red squares marking the corner positions.

The theoretical basis for event generation in neuromorphic vision sensors is derived from the optical flow constraint, which describes the relationship between image intensity changes, spatial gradients, and sensor motion. Specifically, the temporal change in image intensity at a given pixel, I_t , can be expressed as a function of the spatial intensity gradients I_x and I_y and the image-plane velocities V_x and V_y as Equation 5.1

$$I_t = -I_x V_x - I_y V_y \quad (5.1)$$

where I_t denotes the derivative of intensity with respect to time, I_x and I_y are the spatial derivatives of intensity with respect to the x and y coordinates, and V_x and V_y represent the velocities of motion along the x and y axes, respectively. This equation indicates that the temporal change in intensity, and consequently the probability of event generation, is maximized when the spatial gradient of the image (i.e., the edge direction) is perpendicular to the direction of motion. Conversely, when the edge is parallel to the motion, the product of the spatial gradient and velocity is minimized, resulting in few or no events being triggered. Thus, the response of event cameras is intrinsically dependent on the orientation of both the edge and the motion, with the strongest event activity occurring at locations where the motion is orthogonal to image gradients.

This equation explains the observed concentration of events at the corner points in the CIFAR10-DVS recordings. When a frame containing rich edge information moves along a straight path, only edges that are nearly perpendicular to the direction of motion are likely to generate events, while those that are parallel produce very few or none. As a result, during straight-line movement, the event stream is dominated by repeated events triggered by the same set of edges, while many other edges are rarely or never activated. This leads to large portions of the image remaining underrepresented in the event data. At each corner, however, the motion abruptly changes direction, introducing at least two distinct movement vectors. In the case of the diamond trajectory used in CIFAR10-DVS, these vectors are perpendicular, so edges that were previously parallel to the direction of motion-and thus produced few or no events-now become perpendicular to the new motion and start generating a large number of events. This sudden increase fills in the information that is missed during the straight segments. These findings demonstrate that a greater diversity of motion directions is necessary to generate more uniform and complete event data.

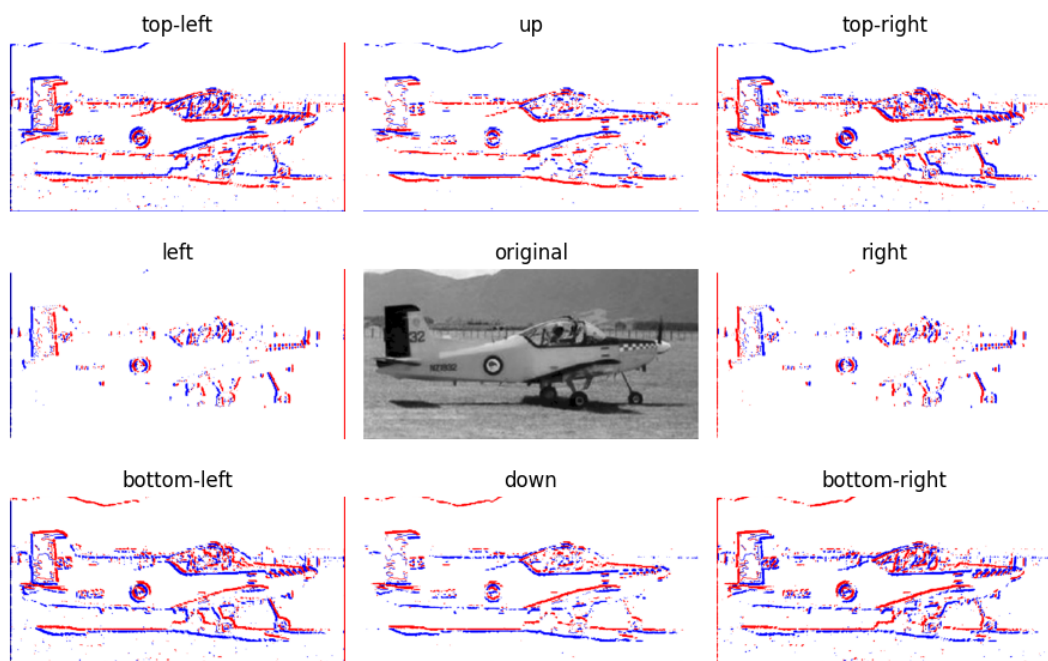


Figure 5.2: Simulation of event generation as a function of motion direction using image gradient information. The central image shows the original frame, while the surrounding panels visualize the simulated event patterns for eight principal motion directions. Positive and negative events are color-coded in red and blue, respectively.

As shown in Figure 5.2, the effect of motion direction on event generation can be observed directly. When the simulated movement is horizontal, almost no events are generated along the airplane's horizontal edges due to the low brightness gradient in the direction of motion. Vertical movement also results in a loss of detail, as many features aligned with the motion

direction are not captured. In contrast, for diagonal movements-such as top-left and bottom-right-the event patterns reveal much more of the airplane's edge structure. These directions combine information from both horizontal and vertical gradients, leading to a denser and more complete event representation of the object. This suggests that motion containing multiple gradient directions can provide richer event data, which is likely to support better classification and recognition performance for the target.

Orchard et al. [36] mentioned that the image velocity observed by a neuromorphic vision sensor is determined by both the translational and rotational motion of the camera relative to the scene. This relationship can be mathematically described as follows:

$$\begin{aligned} V_x &= \frac{T_z x - T_x}{z} - \omega_y + \omega_z y + \omega_x x y - \omega_y x^2 \\ V_y &= \frac{T_z y - T_y}{z} + \omega_x - \omega_z x + \omega_y x y - \omega_x y^2 \end{aligned} \quad (5.2)$$

where V_x and V_y are the velocities in the x and y directions on the image plane, T_x , T_y , and T_z are the translational velocities, ω_x , ω_y , and ω_z are the rotational velocities around the corresponding axes, x and y are the image coordinates, and z is the depth of the scene point relative to the sensor.

From Equation 5.2, it is evident that the terms associated with translational motion, $\frac{T_z x - T_x}{z}$ and $\frac{T_z y - T_y}{z}$, are inversely proportional to the scene depth z . This means that for translational movement, objects at different depths produce different image velocities, resulting in a highly non-uniform event distribution. However, the terms involving rotational motion depend solely on the image coordinates and the angular velocities, and are entirely independent of z . As a result, when the sensor undergoes pure rotational motion, the image velocity for each pixel is determined only by its location in the image plane and the rotation parameters, regardless of how far the object is from the camera. This depth independence enables uniform motion across the entire image, ensuring that event data are distributed evenly throughout the visual field and are not biased by depth variations in the scene.

For these reasons, circular motion-dominated by rotation-allows every part of the image to move with a consistent speed regardless of depth, while the direction of motion is constantly changing. This ensures that edges of all orientations are activated evenly over time, leading to a more balanced and comprehensive distribution of events across the image. The next section discusses how this approach is also supported by observations from biological vision systems.

5.2 Biological Interpretability of Rotational Motion

Event-based cameras inherently draw inspiration from biological visual systems [2], operating similarly to how neurons in the retina respond primarily to changes in brightness rather than static scenes. Consequently, the design of trajectories for event data collection in neuromorphic systems can naturally seek guidance from the subtle movements present in animal eyes.

Biological visual systems, including those in humans, primates, and birds, exhibit continuous miniature eye movements even during periods of stable fixation. These movements primarily comprise three types: microsaccades, tremor, and drift. Microsaccades are rapid, small-amplitude eye movements occurring at intervals during fixation, typically spanning less than two degrees of visual angle and lasting tens of milliseconds [53]. Tremor refers to extremely small and fast oscillations of the eye, characterized by high frequency and low amplitude vibrations occurring continuously, and is hypothesized to maintain sensory neuron activation and prevent complete image fading. Drift represents slow, smooth, and irregular movements between microsaccades, gently shifting the retinal image to refresh visual stimuli [54].

Among these miniature eye movements, microsaccades play a particularly crucial role in visual perception. Research indicates that microsaccades prevent visual fading by periodically shifting the gaze position slightly, thereby refreshing the image on retinal photoreceptors and preventing neural adaptation to stationary stimuli [53]. Moreover, microsaccades are not random movements but rather exhibit directional preferences, potentially influenced by attentional mechanisms, enhancing perception of spatial details and edges oriented differently in the visual field [37]. This selective directional capability enables microsaccades to significantly enhance the detection of visual edges, especially those oriented perpendicularly to the microsaccade direction, as the movement across such edges generates pronounced local intensity changes.

5.3 Recording Platform Design

5.3.1 System Setup

Leveraging the multi-directional and depth-independent properties of circular trajectories, this work introduces a desktop-scale event data acquisition system. As shown in Figure 5.3, the platform integrates a Dobot Magician 6-DoF robotic arm with a custom-designed 3D-printed dual-camera mount. The mount secures a Dynamic and Active-pixel Vision Sensor with 346×260 resolution (DAVIS346) mono event camera and a Raspberry Pi Camera Module 3 at the same horizontal height, positioning their lenses in close proximity to ensure that both cameras observe the scene from nearly identical viewpoints. The robotic arm can be precisely programmed via a Python application programming interface (API) to execute customizable movement paths, including circular trajectories that closely mimic biologically plausible rotational eye movements.

The addition of the Raspberry Pi camera provides several significant advantages over the native DAVIS346 grayscale frames. While the DAVIS346 offers only 346×260 pixel grayscale images, the Raspberry Pi Camera Module 3 is capable of capturing high-resolution frames up to 4608×2592 pixels and supports high-speed acquisition up to 120 FPS at 640×480 resolution. Furthermore, it is equipped with phase detection auto focus (PDAF), a feature not available on event cameras, enabling consistently sharp imaging across a variety of

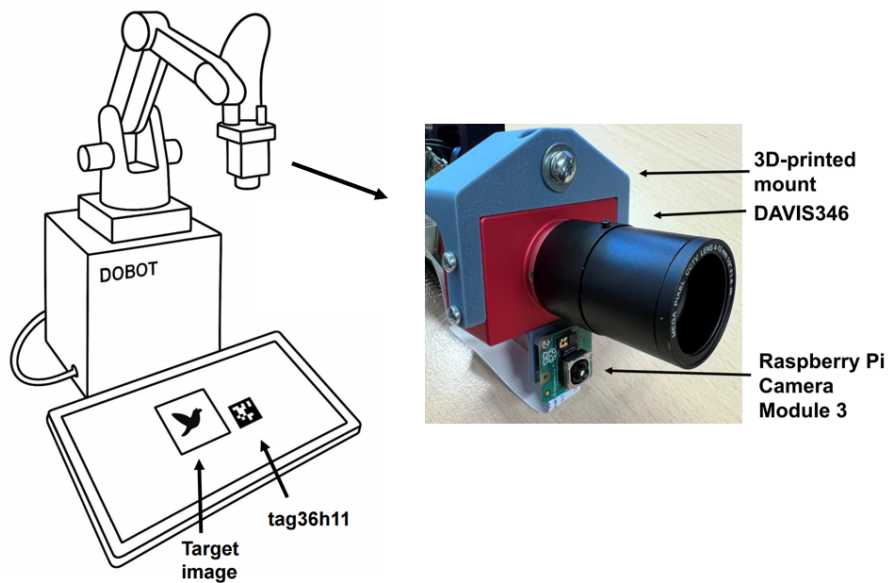


Figure 5.3: Setup of the Dobot, DAVIS346, Pi Camera Module 3, target image and the visual fiducial marker for pose estimation (AprilTag).

experimental conditions.

Integrating a high-resolution red, green, blue (RGB) camera alongside the DAVIS346 event camera in this system provides distinct benefits for the simulation and benchmarking of event-based data. In the field of synthetic event generation from images, researchers have shown that tools such as DVS-Voltmeter and v2e can use high-quality RGB video as input to produce realistic event streams suitable for evaluating models on tasks like segmentation and intensity-image reconstruction [55, 56]. Hybrid datasets, such as EventAid, also rely on synchronized RGB and event camera recordings to support the development and assessment of event-aided image and video enhancement methods [57]. By enabling the collection of temporally aligned RGB and event data from the same perspective, the system established here helps create a versatile resource for both simulation and algorithm validation in event-based vision research.

5.3.2 Time Synchronization

To ensure that frames and events from different devices are precisely aligned in time-despite differences in device clocks and potential timing drift during long experiments-accurate cross-device timestamp alignment is essential. The DAVIS346 sensor records both real event data and grayscale frames, each tagged with Unix time, number of seconds since 1970-01-01 00:00:00 UTC (Unix)-based timestamps that map directly to real-world time and ensure intrinsic internal synchronization. In contrast, the Raspberry Pi camera generates timestamps based on its local system clock, which has no inherent relationship to external or absolute

time, making direct synchronization with the DAVIS346 nontrivial.

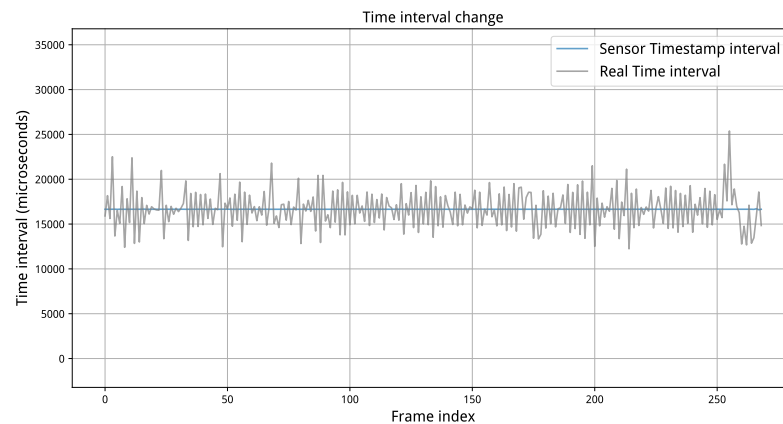


Figure 5.4: Time interval changes over frames.

To address this, we record two types of timestamps for each Raspberry Pi frame: the Sensor-Timestamp, extracted from the camera’s hardware metadata and reflecting the precise moment of exposure; and the RealTime, obtained from the system clock using `datetime.now()`. As illustrated in Figure 5.4, while the RealTime values may exhibit frame-to-frame jitter due to system latency, the mean frame interval remains consistent with that of the Sensor-Timestamp. By analyzing the changes in time intervals between consecutive frames for both SensorTimestamp and RealTime, we identify points where the interval patterns from both clocks closely match. These matching points allow us to estimate the time offset between the two clocks. Once the offset is determined, we can adjust the Raspberry Pi SensorTimestamp to bring it into alignment with the DAVIS346 event timestamps. While this approach cannot eliminate all possible timing discrepancies due to device and system latency, it enables relatively accurate temporal alignment between the two data streams.

This approach operates entirely in software and does not require additional hardware triggers or disruptive light signals, enabling straightforward integration into standard acquisition pipelines. By associating each frame’s hardware-derived SensorTimestamp with its corresponding RealTime value, the method provides continuous, frame-level synchronization and automatic correction for potential clock drift, ensuring robust and stable alignment across devices over extended recording sessions.

5.3.3 ROI Tracking

To track the position of the object of interest as it moves within the field of view due to camera motion, a tag36h11 AprilTag [58] is placed adjacent to the target image. The fixed, known relationship between the tag and the target allows precise estimation of the target’s position, scale, and rotation in every frame. This strategy enables robust region of interest (ROI) throughout the motion sequence, maintaining accuracy even in the presence of mechanical jitter and supporting arbitrary camera trajectories. The trajectory of the tracked

ROI center and its rotation angle over time during circular camera motion can be seen in Figure 5.5.

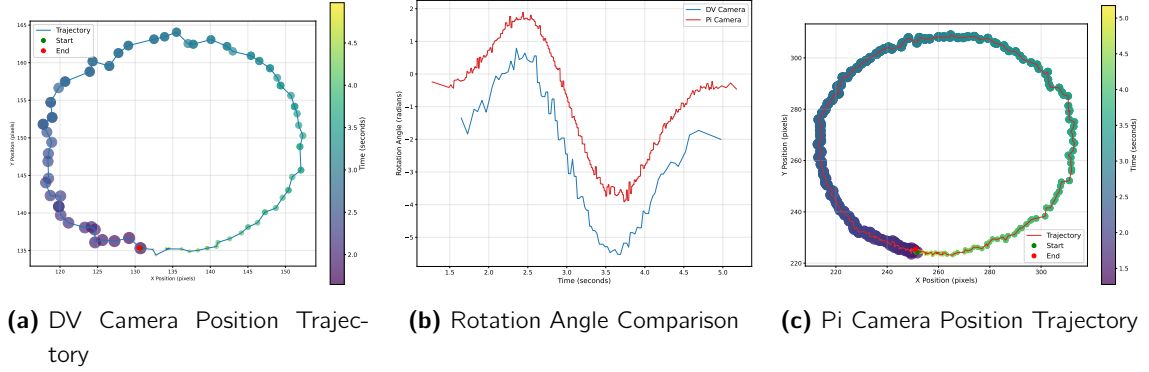


Figure 5.5: Comparison of rotation angle and spatial trajectories between DV and Pi camera data.

Due to the relatively low frame rate of the DV camera's grayscale output, the tracked trajectory for the circular path appears less accurate and exhibits considerable jitter. In contrast, the higher frame rate of the Pi camera yields a smoother and more precise reconstruction of the camera module's circular motion on the robotic arm. This increased temporal resolution not only enhances the fidelity of the tracked spatial trajectory but also provides finer sampling of the rotation angle throughout the motion sequence. Importantly, despite the differences in temporal resolution and imaging modalities, the temporal alignment of the tracked trajectories indicates that the timestamp synchronization between the DV and Pi camera data streams is well maintained, as shown by the close correspondence of their position and rotation profiles.

5.4 Conclusion

As shown in Figure 5.6, a direct comparison between the event data recorded using CIFAR10-DVS and our circular motion system reveals significant differences in both the temporal and frequency domains. In the CIFAR10-DVS data, the event count histogram exhibits periodic, sharp spikes corresponding to trajectory corners, and the FFT displays a dominant high-frequency peak around 20Hz, reflecting repeated abrupt changes in motion direction. In contrast, the data generated by our system present a smooth, bell-shaped event count profile without pronounced peaks, and the FFT is dominated by low-frequency components, with no strong periodic artifacts. This indicates that the circular trajectory ensures continuous, gradual motion, effectively avoiding artificial transitions and resulting in a more stable and consistent event stream.

The primary focus of this work is the development and validation of the event data acquisition system itself. Comprehensive downstream evaluation of the generated static event dataset, such as benchmarking with classification or detection tasks, will be considered in future work.

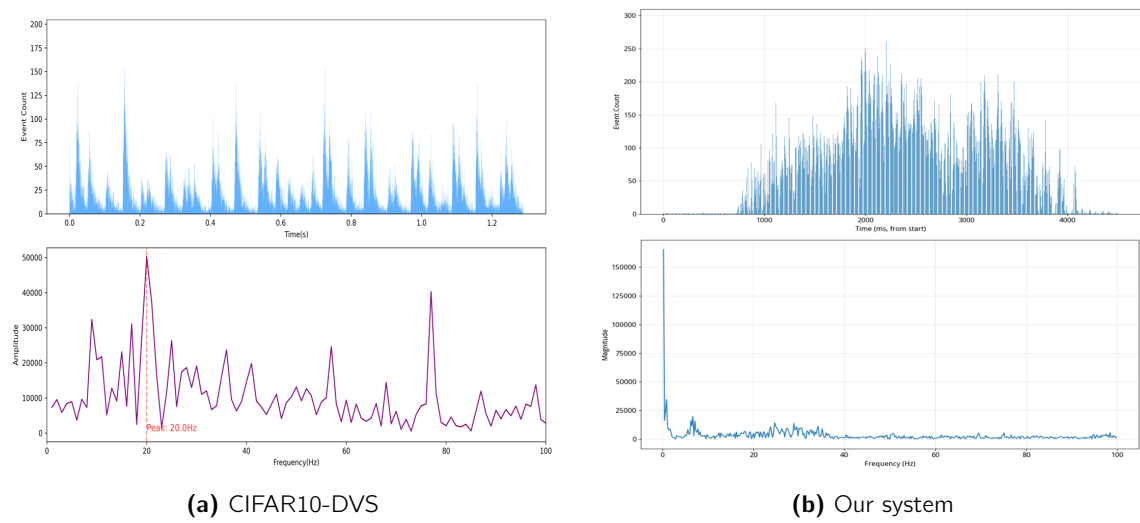


Figure 5.6: Comparison of event data statistics for CIFAR10-DVS and our circular motion system. Each subplot shows the event count histogram (top) and corresponding FFT (bottom).

as the next step of the system's application.

Chapter 6

Conclusion and future work

6.1 End-to-End Event Vision System Framework and Overall Innovations

This thesis presents a complete system for event-based visual recognition, spanning event data acquisition, representation, neural network training, and embedded deployment. Event data are generated using event cameras (DVS), which output asynchronous streams reflecting local brightness changes. To prepare these data for neural networks, several representation methods are compared, with emphasis on maintaining spatial and temporal features while keeping processing efficient.

A co-design approach is adopted, where lightweight CNNs are selected and optimized in parallel with the data representations. Models pre-trained on ImageNet are fine-tuned for event data, with Dropout and learning rate (LR) schedules applied to improve performance on smaller datasets.

For deployment, the system targets FPGA hardware. Neural networks are pruned, quantized, and converted using QAT, then compiled for hardware acceleration. The final models are evaluated on real devices for speed, power, and accuracy.

To improve data quality, a circular motion recording platform is designed, inspired by natural eye movements, producing event streams with smooth distributions and fewer artifacts. This integrated workflow ensures compatibility between event data, neural networks, and embedded hardware.

6.2 Key Experimental Findings and Comparative Analysis

Using a single time window stacking at 128×128 resolution, the proposed model (Figure 3.5) achieves a top-1 accuracy of 81.05% with only 2.4 million parameters. In comparison (Table. 3.4), representative frame-based and voxel-based methods such as HATS and RG-CNNs achieve 52.4% and 54.0%, respectively. Among token-based models, Nested-T achieves

78.1% with 4.2 million parameters, and GET reaches 84.8% with 4.5 million parameters, but both require more complex architectures and higher computational costs. These results highlight that the proposed model achieves competitive accuracy with a smaller model size and a simpler structure compared to existing approaches.

During transfer learning, experiments demonstrated that freezing only the upper layers of the network was not sufficient. Instead, updating all network layers during fine-tuning yielded better adaptation to the event data domain (Table 3.2), indicating that both low-level and high-level features from conventional images required adjustment for event-based inputs. Dropout and L2 were applied to stabilize training and reduce overfitting (Table 3.3), which was especially important given the relatively small dataset size.

Although EfficientNetB0 performed well in terms of accuracy, deployment to embedded hardware using the Xilinx FPGA platform revealed compatibility issues. Specifically, certain layers in the PyTorch implementation, such as data permutation and the Swish (SiLU) activation, could not be mapped to the FPGA's hardware accelerator and required execution on the CPU, resulting in increased inference latency (Figure 4.2). To address this, the architecture was switched to MobileNetV2 implemented in TensorFlow 2, which provided nearly identical classification performance but ensured all operations were compatible with the hardware accelerator, enabling efficient end-to-end inference on the device.

Hardware tests showed that the quantized model achieved 306 frames per second in single-threaded mode and up to 357 frames per second with four threads on the FPGA platform. The final deployed system maintained an accuracy of 81% and operated at a measured power consumption of 1.77 milliwatts (Table 4.1). The use of QAT recovered nearly all the accuracy lost during PTQ, enabling real-time and energy-efficient operation suitable for embedded applications.

The process of event data collection was refined through the development of a biologically inspired circular motion recording platform, which mimics fixational eye movements (microsaccades, tremor, and drift) observed in human vision [37]. Unlike conventional recording methods that use diamond-shaped or segmented trajectories with sharp turning points, the circular motion platform avoids abrupt velocity and direction changes at corners. This design prevents local event bursts and periodic artifacts in the event stream that typically arise from these turning points, and effectively reduces prominent peaks in the frequency spectrum that are otherwise observed at regular intervals (such as 20Hz for CIFAR10-DVS) due to repeated sharp transitions. The resulting event streams exhibit smoother temporal distributions (Figure 5.6) and a more uniform frequency spectrum, reducing spurious peaks by 63% compared to screen-based recording methods. This enhancement ensures consistent spatiotemporal activation patterns, improving the robustness of downstream neural network processing.

6.3 Limitations and Future work

Improving Temporal Generalization through Multi-Scale Window Design As illustrated in Figure 3.6, the training and validation accuracy of the current model depends strongly on access to the full temporal window of each CIFAR10-DVS sample. This reliance on long event sequences may present a bottleneck for real-time recognition in future embedded deployments. When the model is trained using short window inputs, its accuracy on longer validation windows drops as the window length increases, indicating that models trained with short temporal segments have limited capacity to generalize to complete event sequences. Integrating multi-scale window feature fusion could help enhance the model’s generalization across varying temporal window lengths.

Class-Aware Training Strategies for Variable Temporal Contexts Figure 4.6 further shows that the sensitivity to window length differs between categories. Certain classes, such as airplane and cat, can be reliably recognized even with short input windows, suggesting that their key event features emerge early in the temporal sequence. Conversely, categories such as truck and ship achieve low accuracy with short segments, indicating that their distinctive features require longer temporal contexts for reliable detection. For these more challenging classes, specifically training on short windows may help the model learn more generalizable features and improve performance over a wider range of window lengths.

Enhancing Event-Based Learning through Domain-Specific Pretraining As shown in Table 3.2, the classification accuracy improves steadily as more layers of the network are unfrozen during transfer learning from ImageNet. This trend suggests that the low-level features learned from ImageNet, such as edge and texture detectors, do not fully capture the essential characteristics of event-based data. Kim et al. [35] further demonstrated that initializing models with N-ImageNet, a large-scale event-based dataset, yields higher accuracy on CIFAR10-DVS compared to initialization with ImageNet (an increase from 70.36% to 73.72%). Since N-ImageNet provides richer event-specific representations, this approach offers a promising direction for further improving the accuracy of the model proposed in this work.

Towards Structure-Aware and Hardware-Efficient Model Optimization The pruning process in this work is based on the Vitis AI Optimizer tool, which enhances compatibility with embedded hardware but does not necessarily yield the most structure-aware pruning for the network. Recent work by Fu and Yang [59] has shown that in architectures such as MobileNetV2 and EfficientNetB0, certain activation functions contribute to optimization during training but can be removed at inference time without compromising model accuracy. For instance, the core building block of MobileNetV2 is the inverted residual block, which consists of 1×1 convolutions, depthwise convolutions, and residual connections; however,

both depthwise and 1×1 convolutions often exhibit low hardware utilization.

Fu and Yang proposed the DepthShrinker approach, which involves three main steps: identifying and removing redundant activation functions (such as ReLU6) that are unnecessary for inference, merging adjacent convolutional layers (for example, combining 1×1 convolution, depthwise convolution, and another 1×1 convolution into a single dense 3×3 convolution) to reduce data transfer and improve parallelism, and fine-tuning the model to recover accuracy while introducing a small number of additional activation functions to enhance non-linearity. After these optimizations, MobileNetV2 achieved a $3.9\times$ increase in throughput on a Tesla V100 GPU and even higher accuracy compared to methods like MetaPruning (e.g., an improvement of 3.06%). This result indicates that merging, rather than conventional pruning, can simultaneously achieve high accuracy and hardware efficiency, which is particularly suitable for mobile and edge deployment. However, their open-source tools are currently implemented in PyTorch, so further customization and adaptation to TensorFlow 2 are required for deployment on the KV260 platform.

Improving the Motion Platform through Robotic Arm Enhancement The motion limitations of the desktop event data acquisition platform are primarily determined by the capabilities of the Dobot Magician robotic arm. Due to its hardware design, this robotic arm cannot achieve the high circular motion speeds or the small trajectory radii that are possible with industrial-grade motion control systems, restricting the range of motion patterns available for data collection. In practice, the maximum attainable speed dictates how quickly the system can sample a full range of viewing angles for each static object; if the speed is low, the acquisition process becomes time-consuming and generates larger raw data volumes for each sample. For downstream tasks, these hardware constraints lead to two main challenges: the increased dataset size from slow or wide-radius trajectories imposes greater storage and processing requirements, making large-scale experiments and training less practical, while the motion speed-determines how densely multi-directional event data can be captured within a short time frame, which is crucial for real-time embedded inference. As a result, the current motion limits highlighting the need for improved motion control hardware to broaden the applicability of the platform.

The complete source code is available and documented in Appendix A.

Bibliography

- [1] L. K. Sahoo and V. Varadarajan, “Deep learning for autonomous driving systems: Technological innovations, strategic implementations, and business implications - a comprehensive review”, *Complex Engineering Systems*, vol. 5, no. 1, N/A–N/A, Feb. 18, 2025, Publisher: OAE Publishing Inc., issn: ISSN 2770-6249 (Online). doi: 10.20517/ces.2024.83. [Online]. Available: <https://www.oaepublish.com/articles/ces.2024.83> (visited on 05/17/2025).
- [2] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128× 128 120 dB 15 μ s latency asynchronous temporal contrast vision sensor”, *Solid-State Circuits, IEEE Journal of*, vol. 43, pp. 566–576, Mar. 1, 2008. doi: 10.1109/JSSC.2007.914337.
- [3] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, Jan. 1, 2022, issn: 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2020.3008413. arXiv: 1904.08405[cs]. [Online]. Available: <http://arxiv.org/abs/1904.08405> (visited on 05/17/2025).
- [4] A. I. Maqueda, A. Loquercio, G. Gallego, N. Garcia, and D. Scaramuzza, “Event-based vision meets deep learning on steering prediction for self-driving cars”, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 5419–5427. doi: 10.1109/CVPR.2018.00568. arXiv: 1804.01310[cs]. [Online]. Available: <http://arxiv.org/abs/1804.01310> (visited on 12/17/2024).
- [5] D. Gehrig and D. Scaramuzza, “Low-latency automotive vision with event cameras”, *Nature*, vol. 629, no. 8014, pp. 1034–1040, May 2024, Publisher: Nature Publishing Group, issn: 1476-4687. doi: 10.1038/s41586-024-07409-w. [Online]. Available: <https://www.nature.com/articles/s41586-024-07409-w> (visited on 05/17/2025).
- [6] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, *HATS: Histograms of averaged time surfaces for robust event-based object classification*, Mar. 21, 2018. doi: 10.48550/arXiv.1803.07913. arXiv: 1803.07913[cs]. [Online]. Available: <http://arxiv.org/abs/1803.07913> (visited on 12/05/2024).
- [7] Y. Xu, K. Shidqi, G.-J. van Schaik, R. Bilgic, A. Dobrita, S. Wang, R. Meijer, P. Nemhiani, C. Arjmand, P. Martinello, A. Gebregiorgis, S. Hamdioui, P. Detterer, S. Traferro, M. Konijnenburg, K. Vadivel, M. Sifalak, G. Tang, and A. Yousefzadeh, “Optimizing event-based neural networks on digital neuromorphic architecture: A comprehensive design space exploration”, *Frontiers in Neuroscience*, vol. 18, Mar. 28, 2024, Publisher: Frontiers, issn: 1662-453X. doi: 10.3389/fnins.2024.1335422. [Online]. Available: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2024.1335422/full> (visited on 05/17/2025).
- [8] D. Scaramuzza, *Event cameras tutorial*, https://rpg.ifi.uzh.ch/docs/scaramuzza/2019.07.11_Scaramuzza_Event_Cameras_Tutorial.pdf, Tutorial presented at the International Computer Vision Summer School (ICVSS) 2019, Jul. 2019.

- [9] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: An event-stream dataset for object classification", *Frontiers in Neuroscience*, vol. 11, May 2017. doi: 10.3389/fnins.2017.00309.
- [10] Y. Deng, Y. Li, and H. Chen, "AMAE: Adaptive motion-agnostic encoder for event-based object classification", *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4596–4603, Jul. 2020, Conference Name: IEEE Robotics and Automation Letters, issn: 2377-3766. doi: 10.1109/LRA.2020.3002480. [Online]. Available: <https://ieeexplore.ieee.org/document/9116961> (visited on 12/17/2024).
- [11] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, Feb. 2014, Conference Name: IEEE Transactions on Neural Networks and Learning Systems, issn: 2162-2388. doi: 10.1109/TNNLS.2013.2273537. [Online]. Available: <https://ieeexplore.ieee.org/document/6589170> (visited on 12/17/2024).
- [12] A. Zihao Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based optical flow using motion compensation", presented at the Proceedings of the European Conference on Computer Vision (ECCV) Workshops, 2018. [Online]. Available: https://openaccess.thecvf.com/content_eccv_2018_workshops/w36/html/Zhu_Unsupervised_Event-based_Optical_Flow_using_Motion_Compensation_ECCVW_2018_paper.html (visited on 12/17/2024).
- [13] C. Ye, A. Mitrokhin, C. Fermüller, J. A. Yorke, and Y. Aloimonos, "Unsupervised learning of dense optical flow, depth and egomotion with event-based sensors", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2020, pp. 5831–5838. doi: 10.1109/IROS45743.2020.9341224. [Online]. Available: <https://ieeexplore.ieee.org/document/9341224> (visited on 12/17/2024).
- [14] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatzé, and Y. Andreopoulos, "Graph-based spatio-temporal feature learning for neuromorphic vision sensing", *IEEE Transactions on Image Processing*, vol. 29, pp. 9084–9098, 2020, Conference Name: IEEE Transactions on Image Processing, issn: 1941-0042. doi: 10.1109/TIP.2020.3023597. [Online]. Available: <https://ieeexplore.ieee.org/document/9199543> (visited on 12/17/2024).
- [15] Y. Deng, H. Chen, H. Liu, and Y. Li, *A voxel graph CNN for object classification with event cameras*, Apr. 8, 2022. doi: 10.48550/arXiv.2106.00216. arXiv: 2106.00216[cs]. [Online]. Available: <http://arxiv.org/abs/2106.00216> (visited on 12/17/2024).
- [16] F. Gu, W. Sng, T. Taunyazov, and H. Soh, "TactileSGNet: A spiking graph neural network for event-based tactile object recognition", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2020, pp. 9876–9882. doi: 10.1109/IROS45743.2020.9341421. [Online]. Available: <https://ieeexplore.ieee.org/document/9341421/?arnumber=9341421> (visited on 12/17/2024).
- [17] J. Botzheim, T. Obo, and N. Kubota, "Human gesture recognition for robot partners by spiking neural network and classification learning", in *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, Nov. 2012, pp. 1954–1958. doi: 10.1109/SCIS-ISIS.2012.6505305. [Online]. Available: <https://ieeexplore.ieee.org/document/6505305> (visited on 12/17/2024).
- [18] D. Gehrig, A. Loquercio, K. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data", in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, ISSN: 2380-7504, Oct. 2019, pp. 5632–5642. doi: 10.1109/ICCV.2019.00573. [Online]. Available: <https://ieeexplore.ieee.org/document/9009469> (visited on 12/17/2024).
- [19] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, *A differentiable recurrent surface for asynchronous event-based data*, Jul. 31, 2020. doi: 10.48550/arXiv.2001.03455. arXiv: 2001.03455[cs]. [Online]. Available: <http://arxiv.org/abs/2001.03455> (visited on 12/17/2024).

- [20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Dec. 1998. doi: 10.1109/5.726791.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 05/22/2025).
- [22] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385. arXiv: 1512.03385[cs]. [Online]. Available: <http://arxiv.org/abs/1512.03385> (visited on 05/22/2025).
- [23] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "HFirst: A temporal approach to object recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2028–2040, Oct. 2015, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, issn: 1939-3539. doi: 10.1109/TPAMI.2015.2392947. [Online]. Available: <https://ieeexplore.ieee.org/document/7010933/?arnumber=7010933> (visited on 12/17/2024).
- [24] T. Bian, X. Xiao, T. Xu, P. Zhao, W. Huang, Y. Rong, and J. Huang, "Rumor detection on social media with bi-directional graph convolutional networks", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 1, pp. 549–556, Apr. 3, 2020, Number: 01, issn: 2374-3468. doi: 10.1609/aaai.v34i01.5393. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/5393> (visited on 05/22/2025).
- [25] Y. Peng, Y. Zhang, Z. Xiong, X. Sun, and F. Wu, *GET: Group event transformer for event-based vision*, Oct. 4, 2023. doi: 10.48550/arXiv.2310.02642. arXiv: 2310.02642[cs]. [Online]. Available: <http://arxiv.org/abs/2310.02642> (visited on 12/17/2024).
- [26] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, Apr. 10, 2015. doi: 10.48550/arXiv.1409.1556. arXiv: 1409.1556[cs]. [Online]. Available: <http://arxiv.org/abs/1409.1556> (visited on 05/22/2025).
- [27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, 2019. arXiv: 1801.04381 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1801.04381>.
- [28] M. Tan and Q. V. Le, *EfficientNet: Rethinking model scaling for convolutional neural networks*, Sep. 11, 2020. doi: 10.48550/arXiv.1905.11946. arXiv: 1905.11946[cs]. [Online]. Available: <http://arxiv.org/abs/1905.11946> (visited on 05/22/2025).
- [29] Y. Kim, J. H. Baek, I. H. Im, D. H. Lee, M. H. Park, and H. W. Jang, "Two-terminal neuromorphic devices for spiking neural networks: Neurons, synapses, and array integration", *ACS nano*, vol. 18, no. 51, pp. 34 531–34 571, Dec. 24, 2024, issn: 1936-086X. doi: 10.1021/acsnano.4c12884.
- [30] K. Zhong, S. Zeng, W. Hou, G. Dai, Z. Zhu, X. Zhang, S. Xiao, H. Yang, and Y. Wang, "CoGNN: An algorithm-hardware co-design approach to accelerate GNN inference with mini-batch sampling",
- [31] L. Dana, M. S. Pydi, and Y. Chevaleyre, *Memorization in attention-only transformers*, Mar. 10, 2025. doi: 10.48550/arXiv.2411.10115. arXiv: 2411.10115[cs]. [Online]. Available: <http://arxiv.org/abs/2411.10115> (visited on 05/22/2025).
- [32] M. Huh, P. Agrawal, and A. A. Efros, *What makes imagenet good for transfer learning?*, 2016. arXiv: 1608.08614 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1608.08614>.
- [33] Y. Yang, L. Pan, and L. Liu, *Event camera data pre-training*, 2023. arXiv: 2301.01928 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2301.01928>.

- [34] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-dvs and mnist-dvs. their history, how they were made, and other details", *Frontiers in Neuroscience*, vol. 9, Dec. 2015. doi: 10.3389/fnins.2015.00481.
- [35] J. Kim, J. Bae, G. Park, D. Zhang, and Y. M. Kim, *N-imagenet: Towards robust, fine-grained object recognition with event cameras*, 2022. arXiv: 2112.01041 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2112.01041>.
- [36] G. Orchard, A. Jayawant, G. Cohen, and N. Thakor, *Converting static image datasets to spiking neuromorphic datasets using saccades*, 2015. arXiv: 1507.07629 [cs.DB]. [Online]. Available: <https://arxiv.org/abs/1507.07629>.
- [37] R. Engbert, "Microsaccades: A microcosm for research on oculomotor control, attention, and visual perception", *Progress in brain research*, vol. 154, pp. 177–92, Feb. 2006. doi: 10.1016/S0079-6123(06)54009-9.
- [38] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1409.1556>.
- [39] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, *Accurate, large minibatch sgd: Training imagenet in 1 hour*, 2018. arXiv: 1706.02677 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1706.02677>.
- [40] I. Loshchilov and F. Hutter, *Sgdr: Stochastic gradient descent with warm restarts*, 2017. arXiv: 1608.03983 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1608.03983>.
- [41] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, *Bag of tricks for image classification with convolutional neural networks*, 2018. arXiv: 1812.01187 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1812.01187>.
- [42] I. Sadrtudinov, M. Kodryan, E. Pokonechny, E. Lobacheva, and D. Vetrov, *Where do large learning rates lead us?*, 2024. arXiv: 2410.22113 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2410.22113>.
- [43] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatzé, and Y. Andreopoulos, "Graph-based spatio-temporal feature learning for neuromorphic vision sensing", *IEEE Transactions on Image Processing*, vol. 29, pp. 9084–9098, 2020, Conference Name: IEEE Transactions on Image Processing, issn: 1941-0042. doi: 10.1109/TIP.2020.3023597. [Online]. Available: <https://ieeexplore.ieee.org/document/9199543> (visited on 12/17/2024).
- [44] Y. Deng, H. Chen, and Y. Li, "MVF-net: A multi-view fusion network for event-based object classification", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 12, pp. 8275–8284, Dec. 2022, Conference Name: IEEE Transactions on Circuits and Systems for Video Technology, issn: 1558-2205. doi: 10.1109/TCSVT.2021.3073673. [Online]. Available: <https://ieeexplore.ieee.org/document/9406060> (visited on 12/17/2024).
- [45] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, *Direct training for spiking neural networks: Faster, larger, better*, 2018. arXiv: 1809.05793 [cs.NE]. [Online]. Available: <https://arxiv.org/abs/1809.05793>.
- [46] B. Xie, Y. Deng, Z. Shao, H. Liu, and Y.-F. Li, "Vmv-gcn: Volumetric multi-view based graph cnn for event stream classification", *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, Jan. 2022. doi: 10.1109/LRA.2022.3140819.
- [47] Y. Li, H. Zhou, B. Yang, Y. Zhang, Z. Cui, H. Bao, and G. Zhang, *Graph-based asynchronous event processing for rapid object recognition*, 2023. arXiv: 2308.14419 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2308.14419>.
- [48] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, *Incorporating learnable membrane time constant to enhance learning of spiking neural networks*, 2021. arXiv: 2007.05785 [cs.NE]. [Online]. Available: <https://arxiv.org/abs/2007.05785>.

- [49] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, *Swin transformer v2: Scaling up capacity and resolution*, 2022. arXiv: 2111.09883 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2111.09883>.
- [50] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Arik, and T. Pfister, "Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 3417–3425, Jun. 2022. doi: 10.1609/aaai.v36i3.20252.
- [51] AMD Xilinx, *Vitis AI User Guide (UG1414), Version 3.0*, <https://docs.amd.com/r/3.0-English/ug1414-vitis-ai>, Accessed: 2025-05-31, Feb. 24, 2023.
- [52] Z. Ning, M. Vandersteegen, K. Van Beeck, T. Goedemé, and P. Vandewalle, "Power consumption benchmark for embedded ai inference", Oct. 2024.
- [53] S. Martinez-Conde, S. Macknik, X. Troncoso, and T. Dyar, "Microsaccades counteract visual fading during fixation", *Neuron*, vol. 49, pp. 297–305, Feb. 2006. doi: 10.1016/j.neuron.2005.11.033.
- [54] E. Ahissar, A. Arieli, M. Fried, and Y. Bonneh, "On the possible roles of microsaccades and drifts in visual perception", *Vision research*, vol. 118, Dec. 2014. doi: 10.1016/j.visres.2014.12.004.
- [55] S. Lin, Y. Ma, Z. Guo, and B. Wen, "DVS-Voltmeter: Stochastic Process-Based Event Simulator for Dynamic Vision Sensors", in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., Cham: Springer Nature Switzerland, 2022, pp. 578–593, isbn: 978-3-031-20071-7. doi: 10.1007/978-3-031-20071-7_34.
- [56] Y. Hu, S.-C. Liu, and T. Delbruck, *V2e: From Video Frames to Realistic DVS Events*, Apr. 2021. doi: 10.48550/arXiv.2006.07722. arXiv: 2006.07722 [cs]. (visited on 12/12/2024).
- [57] P. Duan, B. Li, Y. Yang, H. Lou, M. Teng, Y. Ma, and B. Shi, *Eventaid: Benchmarking event-aided image/video enhancement algorithms with real-captured hybrid dataset*, 2023. arXiv: 2312.08220 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2312.08220>.
- [58] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4193–4198. doi: 10.1109/IROS.2016.7759617. (visited on 05/02/2025).
- [59] Y. Fu, H. Yang, J. Yuan, M. Li, C. Wan, R. Krishnamoorthi, V. Chandra, and Y. C. Lin, *Depthshrinker: A new compression paradigm towards boosting real-hardware efficiency of compact neural networks*, 2025. arXiv: 2206.00843 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2206.00843>.

Appendix A

Code Repository

The complete source code for this thesis is available on GitHub at:

- <https://github.com/linenmin/Event-Based-Neural-Network-for-Embedded-Deployment>

This repository contains:

- Data representation processing scripts for event data.
- Neural network training code, including data loading and augmentation.
- Deployment utilities covering model inspection, pruning, quantization, and inference for embedded platforms.
- Acquisition system code for timestamp alignment and event data processing based on tag-assisted ROI tracking.

All scripts and documentation are organized in corresponding subfolders for reproducibility and extension.