technische universität dortmund

Lehrstuhl für Förder- und Lagerwesen

Master Thesis

Motion Segmentation in Dynamic Environments using Event Cameras

Rahul Panchal
Matriculation No.: 236510
Course: Automation and Robotics

Issued on:
09.04.2024

Submitted on:
09.10.2024

Examiner:
Prof.'in Dr.-Ing. Alice Kirchheim
M.Sc. Shrutarv Awasthi

TU Dortmund University, Dortmund, Germany
Faculty of Mechanical Engineering
Chair of Material Handling and Warehousing

"This important part of my masters ‘Thesis’ is devoted to my dearest family, especially my solicitous mother and my encouraging father. Their steadfast love, support and unconditional faith have inspired and motivated me throughout my journey. I am deeply indebted to get a mentor like Shrutarv Awasthi who with his ultimate guidance and exponential knowledge has lent me a hand throughout the thesis. His mastery and proficiency have helped me to expand my knowledge and vision about the thesis in a desired way. My dedication is also a tribute to all the striving robotics professionals, all around the globe who share the same enthusiasm towards robotics and constantly make efforts to develop this field for the betterment of the world."

Abstract

This thesis addresses the significant challenge of motion segmentation in dynamic environments, a crucial factor in advancing the capabilities of autonomous systems, including robots and surveillance technologies. The primary aim is to improve the precision and efficiency of distinguishing moving objects from their backgrounds, even in rapidly changing conditions. To achieve this, the study employs state-of-the-art event camera technology, which captures pixel-level brightness changes asynchronously, enabling high temporal resolution data collection. A comprehensive dataset derived from simulations of various motion scenarios is utilized, providing a robust foundation for refining the motion segmentation algorithms. This dataset covers diverse environmental conditions and motion dynamics, which are essential for developing a reliable framework capable of processing complex scene interactions.

The focus of this thesis is the effective extraction of motion information from event camera data. Initially, classical methods such as energy minimization and spatio-temporal graph cuts are explored to establish a solid understanding of motion dynamics. However, as the limitations of these traditional methods become evident—particularly in high-speed or unpredictable scenarios—the study shifts towards a more innovative approach utilizing deep learning techniques. In this work, Convolution Neural Network (CNN), specifically the ResNet architecture, are adapted to process event camera data, leveraging their strengths in feature extraction and motion estimation. The ResNet model, known for its ability to retain fine-grained details through its residual connections, is particularly effective for segmentation tasks, allowing the network to handle complex motion patterns with greater accuracy. This hybrid approach integrates the advantages of classical techniques with the adaptability of CNN-based methods, enhancing the system's performance in real-time applications. Various architectures, including ResNet-based models, are implemented and rigorously evaluated to optimize performance, using a comprehensive set of metrics.

The challenges encountered during the integration of neural networks are analyzed, leading to the development of solutions that improve both accuracy and efficiency. This research contributes valuable insights into the field of motion segmentation, employing cutting-edge methods to address the complexities of real-world environments. The significance of this thesis extends beyond its technical achievements, as the findings offer practical applications in various domains. By delivering a scalable and effective solution, this work enhances the reliability of autonomous systems and lays the groundwork for future advancements in motion analysis and computer vision. Ultimately, this study demonstrates the potential of combining classical approaches with modern machine learning techniques to address critical challenges in vision-based developments utilizing event camera technology.

Acknowledgements

I would like to express my heartfelt gratitude to everyone who contributed to the completion of this thesis and supported me throughout the journey.

First and foremost, I extend my deepest thanks to my professor, Prof.'in Dr.-Ing. Alice Kirchheim, for her unwavering support and guidance. Her expertise and profound knowledge were invaluable in helping me navigate the challenges of this thesis and achieve my goals. She offered insightful feedback, guiding me through mistakes with patience and precision, always pointing me in the right direction. Her encouragement and expertise helped me stay focused and motivated throughout the entire process.

I would also like to express my sincere gratitude to my supervisor, M.Sc. Shrutarv Awasthi, whose expertise and vision greatly enhanced my understanding of the research process. His support and enthusiasm for the project fueled my own passion, and his guidance was instrumental in keeping me focused and motivated throughout.

The journey of this thesis was not without its challenges, but with perseverance and the steadfast support of my professor and supervisor, I was able to expand my knowledge in this field. This experience has been a pivotal part of my Master's program, teaching me not only how to research, articulate, and implement complex ideas but also to grow personally and professionally.

I would like to thank my family and friends for their constant encouragement during this important phase of my life. Their emotional support helped me stay grounded, reminding me of the importance of balance throughout this demanding process. I am also grateful to my co-workers, who were always ready to offer help whenever I needed it.

Finally, I extend my sincere thanks to the Chair of Material Handling & Warehousing (FLW), TU Dortmund University for providing me with this remarkable opportunity. This experience has been an integral part of my academic journey, and I am incredibly grateful for the support I have received.

Contents

1 Introduction **1**
- 1.1 Problem Definition . . . 1
- 1.2 Objectives . . . 2
- 1.3 Approach . . . 2
- 1.4 Tools used . . . 2
- 1.5 Overview . . . 3

2 Fundamentals **4**
- 2.1 Event Cameras . . . 4
  - 2.1.1 Operating principle . . . 4
  - 2.1.2 Event Processing . . . 5
  - 2.1.3 Event data representation . . . 6
  - 2.1.4 Vision based algorithms & applications . . . 6
    - 2.1.4.1 Feature Detection and Tracking . . . 6
    - 2.1.4.2 Optical Flow Estimation . . . 7
    - 2.1.4.3 Pose Estimation & SLAM . . . 8
    - 2.1.4.4 Motion Segmentation . . . 8
- 2.2 Motion Segmentation . . . 9
  - 2.2.1 Definition . . . 9
  - 2.2.2 Motion Segmentation Approaches . . . 9
- 2.3 Space-time Graph Model for segmentation . . . 10
  - 2.3.1 Space-time graphs . . . 10
  - 2.3.2 Delaunay Triangulation . . . 11
  - 2.3.3 Framework for Segmentation . . . 12
    - 2.3.3.1 Markov Random Fields (MRF) . . . 12
    - 2.3.3.2 Energy Minimization . . . 13
- 2.4 Neural Networks . . . 14
  - 2.4.1 Architecture . . . 14
  - 2.4.2 Activation Functions . . . 15
  - 2.4.3 Types of Neural Networks . . . 18
- 2.5 Convolution Neural Networks . . . 20
  - 2.5.1 Structure of a Convolutional Network . . . 20
    - 2.5.1.1 Convolutional Layer . . . 20
    - 2.5.1.2 Pooling Layer . . . 21
    - 2.5.1.3 Fully-connected Layer . . . 22
  - 2.5.2 Data pre-processing . . . 22
- 2.6 U-Net Architecture . . . 23
- 2.7 Transfer Learning with ResNet . . . 24

3 Literature Review **26**

4 Methods **28**
- 4.1 Motion Segmentation using Energy Minimization . . . 28
  - 4.1.1 Problem Formulation . . . 28
  - 4.1.2 Space-Time Graph Model Construction . . . 28
  - 4.1.3 Multi-Model Fitting . . . 29

4.1.4 Energy minimization . . . 30
4.1.5 Optimization . . . 30
4.1.6 Initialization . . . 31
4.2 Neural Network Approach for Depth and Motion Estimation . . . 31
4.2.1 Problem Formulation . . . 31
4.2.2 Data Acquisition & Processing . . . 31
4.2.3 Architectures & Implementation . . . 31
4.2.3.1 Evenly Cascaded Network (ECN) . . . 32
4.2.3.2 Encoder-Decoder Network (EDN) . . . 34
4.2.3.3 ResNet18 Encoder-Decoder Network (REDN) . . . 36
4.2.3.4 Color Scheme for Depth and Motion Outputs . . . 38
4.2.4 Loss Functions . . . 39
4.2.4.1 Photometric Loss . . . 39
4.2.4.2 Explainability Loss . . . 40
4.2.4.3 Smoothness Loss . . . 40
4.2.4.4 Pose Loss . . . 40
4.2.4.5 Combined Loss . . . 40
4.2.5 Best Checkpoint . . . 40
4.2.6 Backward Pass . . . 41
4.2.7 Validation . . . 41
4.2.8 Evaluation . . . 41

5 Datasets **42**
5.1 DVSMOTION20 Dataset . . . 42
5.2 Extreme Event Dataset (EED) . . . 42
5.3 EV-IMO Dataset . . . 43
5.4 Custom Dataset . . . 43

6 Experiments and Results **44**
6.1 Motion Segmentation using Energy Minimization . . . 44
6.1.1 DVSMOTION20 Dataset . . . 44
6.1.2 Extreme Event Dataset (EED) . . . 45
6.1.3 EV-IMO Dataset . . . 45
6.1.4 Custom Dataset . . . 46
6.1.5 Result Analysis . . . 47
6.2 Neural Network Approach for Depth and Motion Estimation . . . 48
6.2.1 Evenly Cascaded Network (ECN) . . . 49
6.2.2 Encoder-Decoder Network (EDN) . . . 52
6.2.3 ResNet18 Encoder-Decoder Network (REDN) . . . 56
6.2.4 Result Analyis . . . 58
6.3 Experimental Summary . . . 59
6.4 Algorithms and Code . . . 59

7 Conclusion & Outlook **60**
7.1 Future Scope . . . 61

1 Introduction

Motion segmentation in autonomous robots, especially in chaotic and highly dynamic environments, is a complex challenge. These robots need to navigate quickly and safely around humans, objects, and other moving entities, all while dealing with the problem of ego-motion—essentially, the movement of the robot's own camera. This motion distorts the images captured, making it harder to segment and track objects accurately. Traditional methods that rely on simplifying assumptions about the scene often fall short in such unpredictable settings, highlighting the need for new, more flexible approaches to tackle these issues effectively [39, 26].

Event cameras offer a cutting-edge solution in vision technology. Unlike standard cameras that capture frames at fixed intervals, event cameras detect changes in brightness at each pixel in real-time, down to the microsecond. This allows them to capture motion with incredible precision and minimal delay, making them perfect for real-time applications like motion analysis [3]. By taking advantage of this technology, event cameras can more accurately and quickly segment moving objects, even in fast-changing and complex environments, revolutionizing fields like computer vision and robotics [28]. Traditional motion segmentation approaches typically rely on handcrafted features or rule-based techniques, such as thresholding and clustering, to find moving objects. While these methods are a good starting point, deep learning models are better at detecting complex patterns in event data, offering more reliable results [25]. Combining both traditional methods and deep learning in hybrid approaches could strike the right balance, using the efficiency of the former and the adaptability of the latter. This master's thesis aims to push forward motion segmentation by blending these strategies, advancing solutions for robotics, surveillance, and autonomous systems.

1.1 Problem Definition

Traditional motion segmentation methods, which rely on fixed assumptions about the environment and scene dynamics, often struggle in unpredictable or complex settings. Event cameras, with their capacity to capture rapid changes in motion at high temporal resolutions, offer a promising alternative. However, existing approaches, based on handcrafted algorithms or deep learning exhibit notable limitations in terms of accuracy and reliability. This thesis aims to investigate a classical technique and a deep learning-based approach to improve motion segmentation in dynamic environments. The goal is to achieve robust and precise segmentation by leveraging the strengths of both methodologies. To this end, the research is structured into the following key tasks:

- Trying out with a classical approach of motion segmentation using spatio-temporal graph cuts. If the method seems inadequate, then shift the focus towards neural network-based solution.
- DVXplorer dataset collection, preprocess the event data by organizing and cleaning it, apply data augmentation techniques to enrich the dataset, and encode the event sequences into a format suitable for input into the neural network model.
- Adaption of the neural network model given in the works of "EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras", and configuration with the DVXplorer event camera dataset.
- Develop a comprehensive training pipeline for the modified neural network model, incorporating tailored loss functions for motion segmentation tasks with event camera data, and establishing training procedures such as hyperparameter tuning and early stopping mechanisms.

- Integrating the trained model into a real-time processing pipeline for event camera data, and testing the modified model on various use cases: first on pre-recorded video data, then on a live feed without ego-motion, and finally on live feed with ego-motion, to assess its adaptability and robustness in different scenarios. Additionally, create a GitHub repository to allow for the reproducibility of the work.

1.2 Objectives

The primary objective of this thesis is to develop a robust framework capable of accurately distinguishing objects from their background in complex and cluttered environments. To achieve this, two distinct approaches have been implemented, providing an indirect visual comparison between the different methodologies. The ultimate goal is to achieve precise and reliable segmentation, ensuring clear separation between objects and background, even in challenging scenarios. Apart from this, the intermediate objectives are as follows:

- Explore and evaluate various approaches to motion segmentation, including both classical techniques and deep learning methods, to determine the most effective solution for different scenarios.
- Understand and implement key data collection and preprocessing techniques necessary for creating a high-quality dataset that enhances model performance.
- Adapt and fine-tune existing neural network architectures for event camera data, experimenting with different hyperparameters to develop an optimized network.
- Integrate appropriate loss functions and training strategies to achieve optimal model performance and accuracy in motion segmentation tasks.
- Implement the developed framework in real-time applications, addressing challenges in autonomous systems and assessing its practical viability in dynamic environments.

1.3 Approach

This thesis is structured in two phases. Initially, a traditional approach based on space-time graphs using energy minimization [48] was implemented and tested across various scenarios to demonstrate its effectiveness. In the second phase, the focus shifted to a data-driven approach, building on existing works [25, 47] to develop and deploy a model. This allowed for a comparison of three different model structures, which were then evaluated on a dataset to identify the most optimal solution for the motion segmentation task.

1.4 Tools used

Several specialized tools and technologies were utilized throughout the research and implementation stages of this thesis, each playing a critical role in the data collection, preprocessing, model development, and evaluation processes. The following are the key tools and software employed:

- **DVXplorer Event Camera:** The DVXplorer event camera was essential for capturing asynchronous changes in pixel brightness, providing high temporal resolution data. This technology was crucial for experimenting with motion segmentation in dynamic and fast-moving environments, see Fig 1.1.

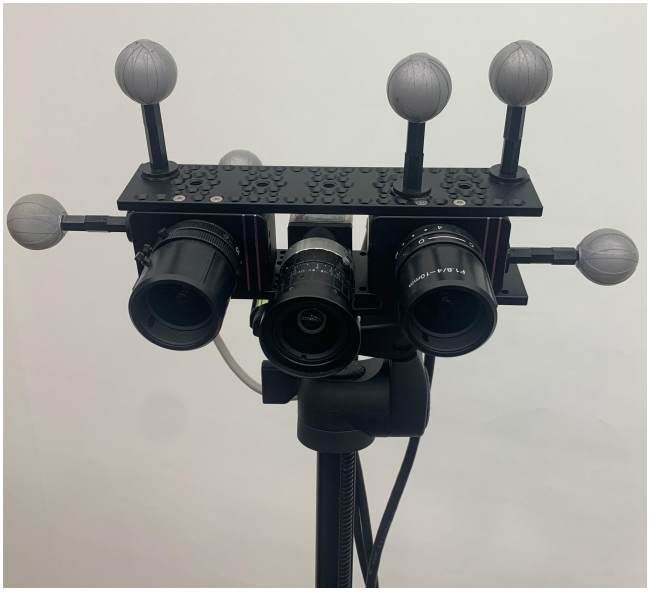


Figure 1.1: DVXplorer Camera for capturing event data [33]

- **Python:** Python was the primary programming language used for developing both classical and neural network-based models.
- **PyTorch:** PyTorch, a widely-used deep learning framework, was employed for building, training, and fine-tuning neural network models
- **OpenCV:** OpenCV, an open-source computer vision library, was used for image and video processing tasks. It facilitated the visualization of event camera data, implementation of optical flow algorithms, and preprocessing of event sequences before feeding them into neural networks for training and evaluation.
- **CUDA and GPU Acceleration:** To expedite the training of deep learning models, GPU acceleration using NVIDIA's CUDA technology was employed. This significantly reduced computation time and enabled efficient handling of large datasets and complex neural network operations.
- **Scikit-learn:** The Scikit-learn library was used for evaluating model performance, providing a variety of machine-learning metrics.

1.5 Overview

This thesis is organized into seven chapters, outlining the implementation of both classical and neural network-based approaches for motion segmentation, with a particular focus on optical flow and CNN. Chapter 2 introduces the fundamental concepts required to understand the work, including discussions on event cameras, classical motion segmentation methods, and an overview of CNN and its architecture. Chapter 3 reviews the latest research and applications in the field of motion segmentation and event cameras, covering both traditional techniques and neural network-based approaches. Chapter 4 presents the methods selected for the motion segmentation task and details their application in this study. Chapter 5 provides a quick overview of the datasets used. Chapter 6 presents the experimental results, offering an analysis of the performance of the different approaches.

Finally, Chapter 7 offers conclusions and future outlook, assessing the performance of the developed models, discussing their limitations, and identifying areas for further research. It also provides a concise summary of the project's results and their significance in achieving the research objectives.

2 Fundamentals

This chapter explores the fundamental aspects of motion segmentation using event cameras, with a particular emphasis on integrating convolutional neural networks. We will explore the operating principle behind event cameras, event-based vision principles, and the convolutional neural network techniques that facilitate robust motion segmentation. We will also examine a few classical approaches to the motion segmentation problem, which have established a foundation for understanding and addressing the complexities of this field. These traditional methods have paved the way for the development of modern deep-learning techniques. Building on this foundation, we will delve deeper into these concepts in the subsequent chapters, focusing on their application to segmenting moving objects in an environment using event cameras.

2.1 Event Cameras

Event cameras, such as the Dynamic Vision Sensor (DVS), Dynamic and Active-pixel Vision Sensor (DAVIS), and DVXplorer, represent a new class of imaging sensors that capture light in a fundamentally different way from traditional frame-based cameras. Instead of recording frames, they detect changes in light intensity at each pixel, producing a sparse stream of data. Each pixel adapts its sampling rate based on the rate of change in the logarithmic intensity signal, meaning that faster motion generates more events per second. These sensors are highly responsive to visual stimuli, transmitting events with sub-millisecond latency and microsecond-level timestamping [48].

Event cameras offer several key advantages over standard cameras, such as low latency, low power consumption, high temporal resolution, and a high dynamic range (HDR). They can rapidly detect brightness changes in analog circuitry and have a 1 MHz clock, enabling the capture of fast motions without blur. Unlike traditional cameras, each pixel in an event camera functions independently, eliminating the need for global exposure time. Event cameras consume minimal power, with most using about 10 mW at the die level, and some prototypes achieving less than 10μW. Embedded event-camera systems have demonstrated system-level power consumption of less than 100 mW, with sensors directly interfacing with processors. The high dynamic range of event cameras ($>$ 120 dB) far exceeds the 60 dB of high-quality frame-based cameras, allowing them to capture images across a wide range of lighting conditions, from moonlight to daylight. This is made possible by the logarithmic response of photoreceptors and the independent operation of each pixel, adapting to both dark and bright stimuli [12].

2.1.1 Operating principle

To understand these advantages more deeply, it is essential to look at the operating principles of event cameras. Event cameras capture visual information in a completely different way than the standard RGB cameras. Unlike the standard cameras that produce frames at fixed intervals, event cameras are responsive to changes in brightness with microsecond latency. They react to changes in light intensity by generating events. This unique characteristic enables them to transmit data only when there is a significant change in the scene's illumination, eliminating redundancy during periods of no motion.

For instance, when observing a spinning disk with a black circle, it memorizes the log intensity and keeps an

eye out for any changes that deviate from this value, by a significant amount. When the change exceeds a certain threshold, the camera transmits an event from the chip with its x, y location co-ordinates, timestamp t (measured in microseconds) of the intensity change, and the 1-bit polarity p of the change (i.e., increase or decrease in the brightness level of the pixel), depicted in Fig. 2.1. This approach results in a higher event rate during rapid signal changes, illustrating their efficiency in capturing dynamic scenes with minimal data output [38, 33].

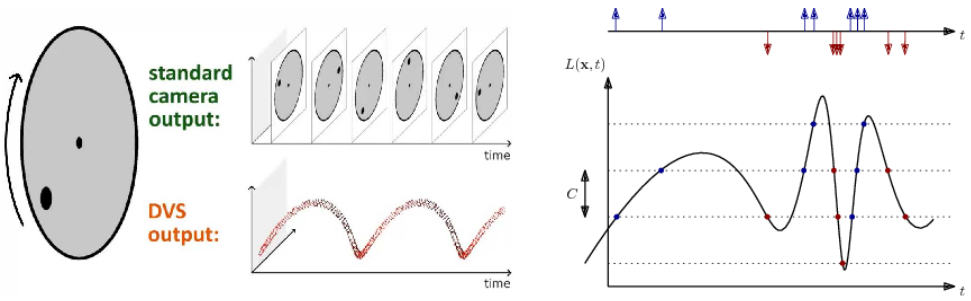


Figure 2.1: Comparison of a standard and an event camera's viewing [33]

The intensity of light at a given pixel is influenced by both the scene's illumination and the surface's reflectance. When the illumination remains relatively constant, a change in the logarithm of light intensity primarily indicates a change in surface reflectance. These variations in reflectance are typically caused by the movement of objects within the field of view. An event camera's pixels operate independently, reacting to variations in their logarithmic photocurrent $L = \log(I)$ ("brightness"). In an ideal, noise-free environment, an event $e_k = (x_k, t_k, P_k)$ is generated at a pixel $x = (x_k, y_k)$ and time t_k whenever a change in brightness occurs since the last event at that particular pixel [12], i.e.,

$$\Delta L(x_k, t_k) = L(x_k, t_k) - L(x_k, t_k - \Delta t_k), \tag{2.1}$$

reaches a temporal contrast threshold C, i.e.,

$$\Delta L(x_k, t_k) = P_k C, \tag{2.2}$$

where C $>$ 0, Δt_k represents the time that has passed since the same pixel at the previous event, and the polarity $p_k \in \{+1, -1\}$ indicates the direction of the brightness change. Hence, each pixel has its sampling rate (which depends on the visual input) and outputs data proportionally to the amount of motion or illumination variations in the scene. An event-based camera records sparse, asynchronous events in the space-time domain instead of continuously producing images.

2.1.2 Event Processing

Event cameras represent a significant shift in how visual information is processed, focusing on extracting meaningful data to address specific tasks. This process is highly dependent on the application, influencing the design of algorithms used for solving these tasks. Event cameras capture data asynchronously and sparsely, offering high temporal resolution and low latency. Two main types of algorithms are used with event cameras: those that process data event-by-event for minimal latency, and those that work with groups or batches of events, which may introduce some latency. Even in the latter, state updates can occur with each new event if the processing window shifts by one event. The difference between these approaches is subtle since an individual event lacks sufficient information for accurate estimation. Algorithms can also be categorized into model-based or model-free approaches, depending on how the events are processed. Additionally, methods are classified by the type of objective or loss function they use, such as geometric, temporal, or photometric-based optimization frameworks [12].

2.1.3 Event data representation

Event data is processed and transformed into various representations to extract meaningful information for specific tasks. These representations are designed to aggregate information from individual events without requiring additional contextual knowledge. Some methods involve straightforward, hand-crafted data transformations, while others employ more sophisticated techniques [12]. These diverse representations serve as the foundation for a wide range of applications, making it essential to thoroughly understand how to transform and utilize event data effectively. In this work, we will primarily focus on individual events and event packets, as these representations enable easier implementation and align closely with our methodology for efficient motion segmentation. However, it is important to recognize that other representations, such as 3D point sets and voxel grids, also offer valuable approaches depending on the specific application.

- **Individual events**
  Individual events represented as, $e_k \doteq (x_k, t_k, p_k)$ signify discrete changes in pixel brightness that occur at specific spatial coordinates x_k and timestamps t_k accompanied by their corresponding polarity p_k. Various methodologies are employed to process these events on a one-by-one basis, utilizing techniques such as Spiking Neural Networks (SNN)s and probabilistic filters. These approaches leverage data accumulated from prior events alongside newly received information, allowing for the asynchronous generation of meaningful outputs.
- **Event Packet**
  Events $\epsilon = \{e_k\}_{k=1}^{N_e}$ occurring within the same spatio-temporal neighbourhood are collectively processed to generate a result. This method maintains accurate information about timestamps and polarities. Selecting the correct packet size, denoted as Ne, is crucial to meet the algorithm's assumptions, such as maintaining a constant speed of motion throughout the packet's duration. The choice of packet size can vary depending on the specific task.

2.1.4 Vision based algorithms & applications

The following section will explore a few vision-based algorithms and their practical implementations, with a particular emphasis on optical flow estimation and motion segmentation, which is the central focus of this work using event cameras. By showcasing the effective application of these representations, this section aims to illustrate their potential use cases in the field of perception.

2.1.4.1 Feature Detection and Tracking

Feature detection and tracking using event cameras utilize the unique properties of these cameras, such as high temporal resolution and resilience to motion blur, to achieve robust and low-latency tracking in dynamic environments. The method begins with detecting features in a reference frame, extracting an image patch around each feature, and using an asynchronous event stream to track the features over time, as illustrated in Fig 2.2. Events are converted into a dense representation for efficient processing by a neural network, which predicts feature displacements and updates their locations iteratively. A frame attention module aggregates information across multiple feature tracks, enhancing robustness. Additionally, a self-supervision strategy allows the network to adapt to different cameras and scenarios without extensive parameter tuning, significantly improving performance in challenging environments [24].

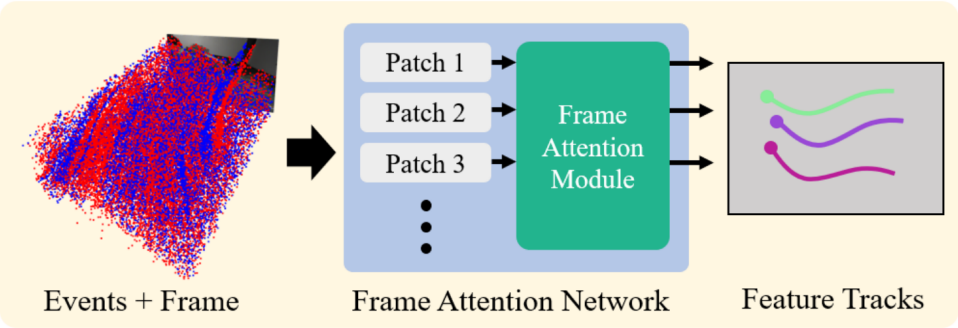


Figure 2.2: Feature detection and tracking using event camera data, highlighting the process of extracting significant features for a frame attention network, which ensures robust tracking across dynamic scenes [24].

2.1.4.2 Optical Flow Estimation

Optical flow estimation from event-based cameras involves processing asynchronous changes in brightness, i.e. events, which occur independently at each pixel. Each event includes precise information about its location within the image, the polarity of the brightness change (increase or decrease), and the exact timestamp of occurrence. To compute optical flow, algorithms integrate these sparse events over time. This integration process aggregates the event data to infer the direction and speed of motion at each pixel. Techniques such as accumulation and interpolation are commonly used to estimate continuous motion vectors across the image plane. This method is particularly beneficial for applications requiring high temporal resolution and low latency, enabling real-time tracking of fast-moving objects and adaptation to dynamic lighting conditions effectively. By leveraging the unique characteristics of event-based data, optical flow estimation provides robustness and accuracy.

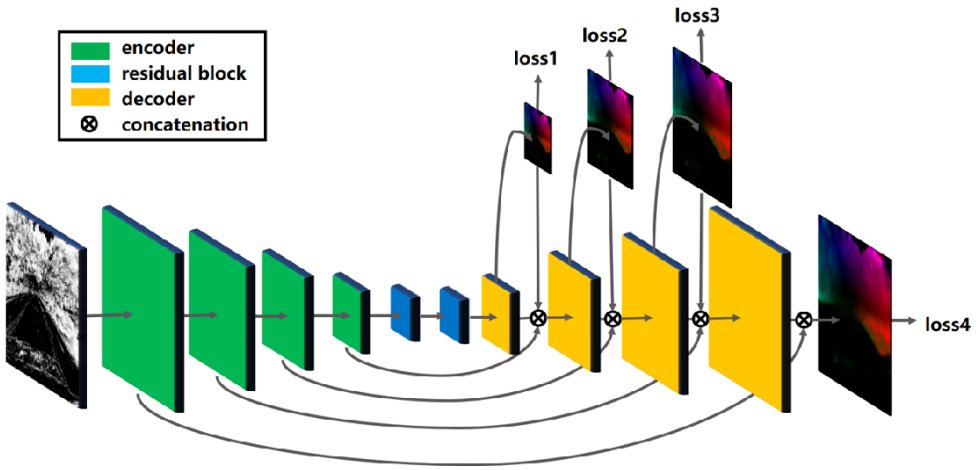


Figure 2.3: Estimation of optical flow using the EV-FlowNet framework, which adeptly captures motion dynamics in event-based data, enhancing both accuracy and performance [49].

Building on these principles, optical flow estimation for event-based cameras can be implemented using neural network architectures specifically designed to manage asynchronous event inputs effectively [49], as shown in Fig. 2.3. Beginning with encoder layers utilizing strided convolutions, this approach extracts hierarchical features and downsamples data. Residual blocks then enhance representation learning while preserving information integrity. Decoding utilizes upsampling convolutional layers with skip connections to merge high-level semantic context and detailed spatial information. Each decoder layer incorporates depthwise convolutions for flow predictions at multiple resolutions, continuously refined through loss computation.

2.1.4.3 Pose Estimation & SLAM

Pose estimation involves determining the camera's position and orientation in 3D space by analyzing the asynchronous stream of events, typically using probabilistic filtering techniques like Bayesian filters to refine the camera's pose estimation based on detected intensity changes over time. Bayesian filters are essential in managing the asynchronous nature of event data and accurately estimating the system state, including the camera's pose and scene structure. They continuously update state estimates based on event probabilities relative to the current system state, overcoming the challenge of sparse event information for estimating parameters such as the camera's six degrees of freedom (6-DOF) pose. Event-based SLAM extends this capability to simultaneously map the environment and track the camera's pose, generating semi-dense maps primarily composed of scene edges triggered by intensity changes, as shown in Fig. 2.4. This map representation aligns with event cameras detecting changes in intensity edges rather than capturing entire image frames [12].

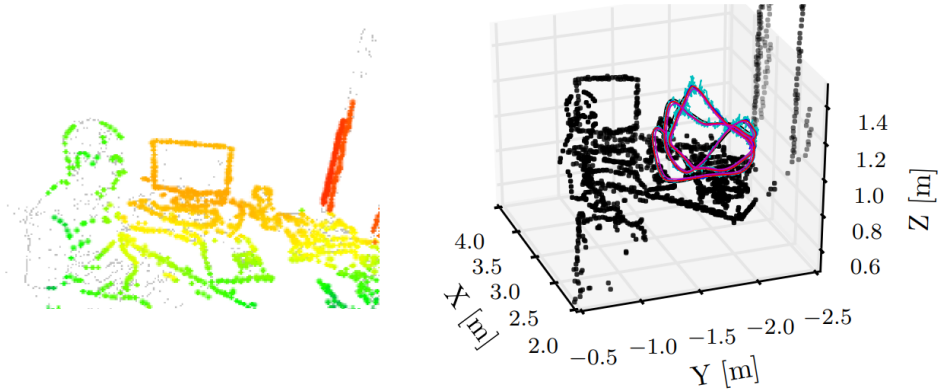


Figure 2.4: The reconstructed scene displays a semi-dense map coloured by depth, overlaid with events in grey, demonstrating excellent alignment between the map and events. The estimated camera trajectory from various methods and the semi-dense 3D map (point cloud) are also depicted [12]

.

2.1.4.4 Motion Segmentation

Motion segmentation using event cameras involves distinguishing moving objects from the background by analyzing asynchronous brightness changes captured as events. This process is straightforward with a stationary event camera, as events are solely attributable to object motion under constant illumination. However, challenges arise with a moving camera, where events are triggered across the entire image plane due to both moving objects and the static scene influenced by the camera's ego-motion. Each event provides minimal information, making per-event classification difficult. Techniques typically involve clustering events based on motion models, such as linear motion hypotheses, to associate events with specific moving objects. Motion-compensated event images associate events that form sharp edges when warped according to motion hypotheses, fitting linear models to dominant events and refining segmentation iteratively. Learning-based approaches employ artificial neural networks to estimate depth, ego-motion, segmentation masks, and object velocities from events, supported by an event-based dataset with accurate motion masks for supervised learning [12, 48].

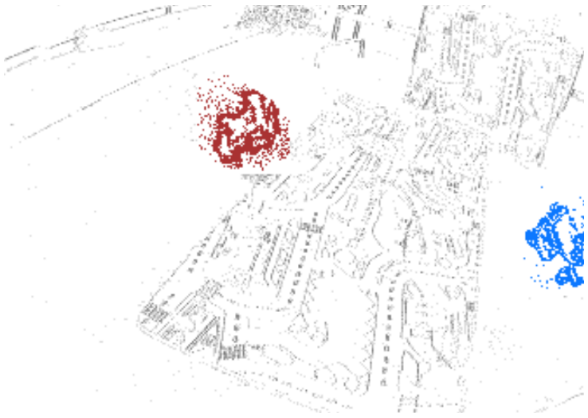


Figure 2.5: Segmentation results on the EVIMO dataset: Gray is used for the background and dark red/blue for moving objects [48]

.

The accurate identification of independently moving objects holds significant importance in comprehending dynamic scenes. However, conventional cameras often encounter challenges related to motion blur and exposure artifacts. To address this, one methodology is to approach the motion segmentation problem as an energy minimization task, involving the fitting of multiple motion models to the data [48]. This optimization leverages the spatio-temporal graph structure of the input event data, enabling iterative solutions for event cluster assignment (labelling) and motion model fitting, to segment moving objects in a dynamic environment, as shown in Fig. 2.5. Since this topic is the primary focus of this thesis, it will be explored in greater detail in the upcoming section.

2.2 Motion Segmentation

This thesis centres on motion segmentation, the main component within the realm of vision-based algorithms. This specialized task plays a crucial role in differentiating moving objects from their backgrounds or other stationary objects, regardless of whether the environment is dynamic or static.

2.2.1 Definition

Motion Segmentation, in the context of traditional cameras, is a task that groups pixels sharing the same motion. Clustering these pixels divides the foreground objects from the background because of different motion types [21]. It is an essential pre-processing task for various computer vision applications, including object tracking, autonomous navigation, activity recognition, and video surveillance, where the primary goal is to separate an object in motion from its background or different objects. It involves dividing an image (from a sequence of images) into various objects, each moving uniquely. One way to perform this is by tracking the points of interest on these objects over a series of images and creating trajectories for each point. By studying these trajectories, we can determine how to segment the image.

2.2.2 Motion Segmentation Approaches

The classical approaches for motion segmentation mainly rely on mathematical and algorithmic solutions to analyse motion within video sequences. Motion information is usually obtained by matching pixels across consecutive frames. This could be in the form of either optical or point trajectories. Several fundamental approaches have been thoroughly investigated in [10] to tackle the motion segmentation problem: one based on motion trajectories, which clusters moving points to distinguish between different objects; another based on projective geometry, which uses the geometric properties of projections to separate moving objects from the background; a third approach based on perspective projection, which analyzes changes in the appearance

of objects due to varying viewpoints; and a modern approach based on convolutional neural networks, which leverages deep learning to learn automatically and segment motion patterns from large datasets.

Several popular and state-of-the-art motion segmentation methods exist, categorized based on their underlying principles. However, some of these methods overlap across multiple categories [23]. A few of the methods used in this study are discussed:

- **Optical Flow:**
  Optical flow detects motion within each pixel neighbourhood by analysing pixel colour changes and intensity between consecutive frames. It generates vectors that represent the direction and magnitude of this motion. These vectors are grouped based on their similarity, enabling the segmentation of distinct moving regions. The primary purpose of optical flow is to segment and isolate individual moving objects within a video. However, optical flow alone is insufficient for motion segmentation, as it struggles with occlusions and temporary stops of objects. It is also sensitive to noise and changes in lighting conditions. Therefore, additional procedures are required along with it to recover accurate boundaries, such as [36] which focuses on using semantic segmentation to enhance optical flow estimation by applying object class labels to determine appropriate motion models, achieving significant improvements in motion and segmentation accuracy. The estimation of optical flow, in conjunction with neural networks, is a central focus of this thesis. The details of this integrated approach will be discussed in Chapter 4.
- **Deep Learning**
  Motion segmentation has been transformed by deep learning. Deep convolutional neural networks (CNNs) can immediately learn motion patterns from video data, in contrast to conventional techniques that depend on manually created features and scene assumptions. This allows them to manage intricate scenes with numerous moving objects and different kinds of motion. By performing joint learning of motion and object features, architectures such as SMSnet [44] streamline the pipeline and achieve state-of-the-art performance. Further pushing the envelope, recent developments such as the use of Graph Transformer Neural Networks (GTNNs) for event-based cameras [4] allow for generalizable motion segmentation, even for unusual sensor data, without requiring a great deal of pre-processing.
- **Graph Based Segmentation** A graph-based approach, using energy minimization techniques like graph cuts and MRFs, offers an effective solution for motion segmentation. The fundamentals of this method are discussed in the Sec. 2.3, and later on the method is elaborated in 4.

2.3 Space-time Graph Model for segmentation

This section introduces space-time graph topologies, highlighting their ability to model relationships between data points such as pixels, features, or frames. A graph-based approach offers a powerful solution to motion segmentation problems, and to further achieve optimal segmentation, energy minimization techniques like graph cuts and Markov Random Field (MRF)s are employed, which will be further discussed in this section.

2.3.1 Space-time graphs

Space-time diagrams, often referred to as Minkowski diagrams, are crucial tools in the study of relativistic physics, providing a visual framework for understanding the relationship between time and space. These diagrams inherently represent four-dimensional space-time, incorporating three spatial dimensions and one temporal dimension. However, since it is not possible to represent four-dimensional space-time (x, y, z, t) on a two-dimensional surface, the three spatial dimensions (x, y, z) are collapsed into a single spatial dimension, while time is treated as a distinct axis. This simplification results in a space-time diagram, as shown in Fig. 2.6, where each specific point in space-time is referred to as a world point or event, and a continuous series of these points that illustrates the motion of an object is called a world line [41].

The world line in a space-time diagram represents an object's trajectory through both space and time. If the object is stationary, its world line appears as a straight vertical line, indicating no change in spatial position over time. In contrast, a moving object will have a sloped world line, with the slope corresponding to the object's velocity. Steeper slopes represent slower speeds, while a world line tilted at a 45° angle reflects motion at the speed of light. If the object accelerates or decelerates, the world line curves, illustrating changes in velocity

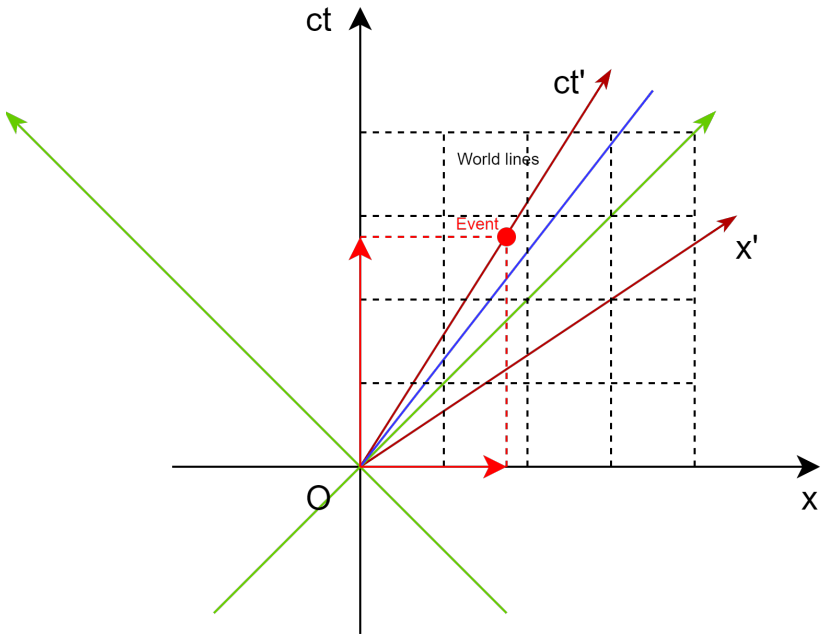


Figure 2.6: ct denotes the time axis and x represents the spatial axis (a line). The diagram is a plane, with light rays traveling at $45°$ and $135°$ angles to the spatial axis (these rays form a two-dimensional light cone). The blue line represents the world line for accelerated motion, while the red line represents the world line of constant motion [41].

over time. Thus, objects moving more slowly exhibit a steeper gradient, while faster-moving objects exhibit a shallower gradient [41].

These diagrams not only clarify the relationship between spatial and temporal dimensions but also provide essential insights into phenomena such as time dilation—the slowing down of time for an object moving at high speeds relative to an observer, length contraction—the shortening of an object's length in the direction of motion as its speed approaches the speed of light, and the relativity of simultaneity—the idea that events which occur simultaneously in one reference frame may occur at different times in another moving reference frame [41].

2.3.2 Delaunay Triangulation

Delaunay triangulation is a geometric method for connecting a set of points in a plane to form a network of triangles. This triangulation maximizes the minimum angle of the triangles, helping to avoid skinny triangles and creating a more stable mesh. A key property of Delaunay triangulation is that for each triangle, the circumcircle does not contain any other points from the set within it, as shown in Fig. 2.7.

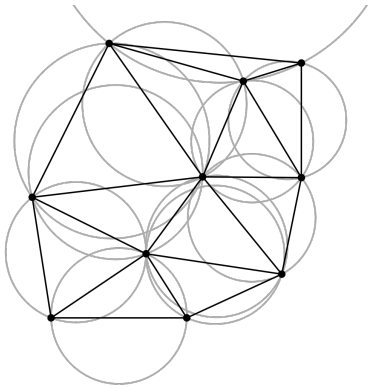


Figure 2.7: Delauanay triangulation, with circumcircles [7]

This technique is instrumental in the construction of space-time graphs, as it allows for the efficient representation of dynamic environments by connecting spatial points over time with well-formed triangular meshes.

2.3.3 Framework for Segmentation

This section discusses Markov Random Fields and energy minimization strategies, which are utilized to improve the effectiveness of image segmentation.

2.3.3.1 Markov Random Fields (MRF)

Before delving into image segmentation via energy minimization, it is essential to understand MRF. MRF is a framework for modelling the joint distribution of an undirected, connected graph. In this context, each node represents a random variable, while each edge indicates a dependency between connected nodes. This structure helps to represent complex relationships and interactions within the data, making it easier to incorporate spatial and contextual information in various applications, such as image segmentation. By using MRFs, we can effectively capture the dependencies between neighbouring variables, which enhances the modelling of uncertainties and improves the accuracy of the segmentation process.

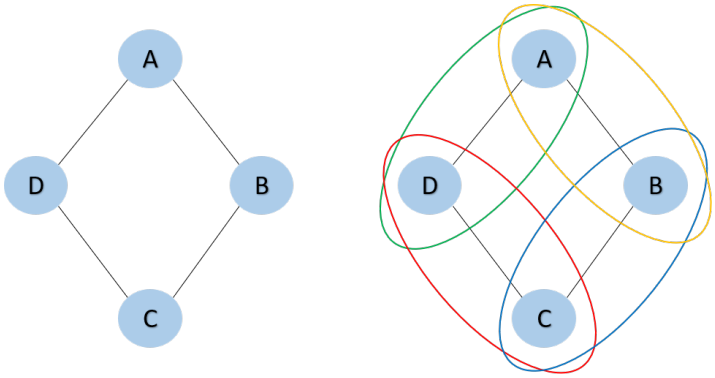


Figure 2.8: Undirected graph, as a joint distribution using MRF [20]

Fig. 2.8 illustrates a graph comprising four nodes: A, B, C, and D, each representing a random variable. Each node can assume s state, which is either 0 or 1. It outlines how the states of these nodes interact, emphasizing the probabilistic connections inherent in the MRF framework

$$
\begin{aligned}
p(A,B,C,D) &= \frac{1}{Z}\tilde{p}(A,B,C,D), \text{ where} \\
\tilde{p}(A,B,C,D) &= \phi(A,B)\phi(B,C)\phi(C,D)\phi(D,A), \\
\phi(X,Y) &= \begin{cases} 10 & \text{if } X=Y=1 \\ 5 & \text{if } X=Y=0 \\ 1 & \text{otherwise.} \end{cases} \\
Z &= \sum_{A,B,C,D} \tilde{p}(A,B,C,D)
\end{aligned}
\tag{2.3}
$$

The joint distribution p represented by the MRF captures the relationships and dependencies between the random variables. The potential function $\phi : X, Y \rightarrow \mathbb{R}$ assigns a weight to the edge (X, Y) based on the states of the random variables X and Y. The weight is higher when both variables take the value 1, compared to when both are 0 or when they differ. This implies that a larger weight assigned to an edge indicates a greater likelihood of the connected random variables sharing the same state. The joint distribution $\tilde{p}$ is defined as the product of the potential functions $\phi(X, Y)$ for all the edges $((AB), (BC), (CD), (DA))$. Z is the constant term, to ensure that the sum of all the probabilities in p must be equal to 1.

2.3.3.2 Energy Minimization

Energy minimization is a foundational approach in image segmentation that employs optimization techniques to achieve coherent and precise results. This method typically involves formulating an energy function that incorporates key elements, such as data fidelity, smoothness constraints, and model complexity. By carefully balancing these factors, energy minimization produces meaningful segmentation that accurately delineates objects within images. This versatile technique is widely utilized across various segmentation tasks, highlighting its effectiveness in enhancing the clarity and accuracy of visual data interpretation [48].

The energy function E used in our work is typically decomposed into three main components:

1. **Data Term:** This term measures how well the segmentation aligns with the observed data. It assesses the discrepancy between the segmented image and the original image, encouraging the segmented regions to be homogeneous:

$$E_{\text{data}}(S) = \sum_i D(I_i, S_i)$$

where S denotes the segmentation map, I_i is the pixel intensity at location i, and D quantifies the difference between the observed intensity and the expected intensity for the segment.

2. **Smoothness Term:** This term enforces spatial coherence by penalizing abrupt changes in segmentation, encouraging neighbouring pixels to belong to the same segment unless strong evidence suggests otherwise:

$$E_{\text{smooth}}(S) = \sum_{(i,j)\in N} V(S_i, S_j)$$

where N represents the set of neighbouring pixel pairs, and V measures the cost associated with differences between neighbouring segment labels.

3. **Minimum Description Length (MDL)**: The MDL principle adds a penalty for model complexity, ensuring that the segmentation is not overly complex. The idea is to prefer simpler segmentations that can be described with fewer bits, thus avoiding overfitting. This component can be expressed as:

$$E_{\text{MDL}}(S) = C(S)$$

where $C(S)$ represents the complexity of the segmentation map S. This term discourages unnecessary subdivisions in the image by favouring simpler models that can be described with minimal parameters.

The total energy $E(S)$ for the segmentation is then formulated as:

$$E(S) = E_{\text{data}}(S) + \lambda_1 E_{\text{smooth}}(S) + \lambda_2 E_{\text{MDL}}(S)$$

where λ_1 and λ_2 are balancing parameters that control the trade-offs between data fidelity, smoothness, and model complexity. To find the optimal segmentation, one seeks to minimize the total energy $E(S)$.

2.4 Neural Networks

Following the examination of various motion segmentation approaches, the discussion now moves to the fundamentals of neural networks, including their basic principles and different types. This conceptual groundwork is essential for comprehending the advanced application of neural networks in motion segmentation addressed in this thesis.

2.4.1 Architecture

A neural network architecture is a basic framework that allows machines to learn from data, identify patterns, and make decisions in a way that's similar to the thought process of humans. As illustrated in Fig. 2.9 this architecture typically consists of interconnected layers of computational units known as $neurons$ or $nodes$, which are systematically organized into three distinct layers: $input$, $hidden$, and $output$, as depicted in Fig. 2.10. The flow of information through these neurons is directed by $connections$ and $weights$, with the dynamics of this process governed by a $learningrule$. Additionally, $activationfunctions$ are critical for introducing non-linearity into the network's computations, thereby enhancing the model's ability to capture complex relationships within the data [2].

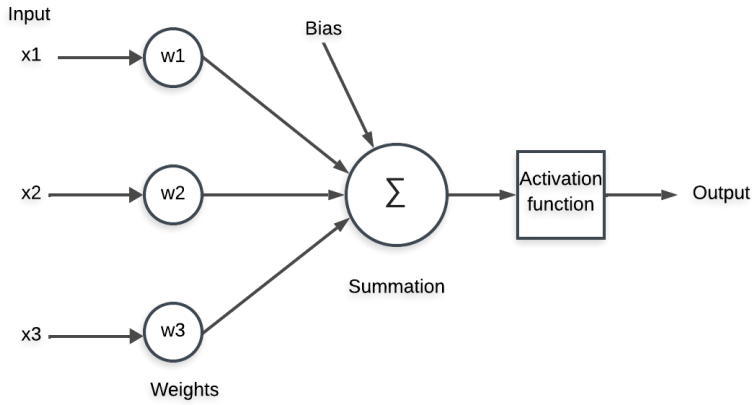


Figure 2.9: Neuron in a neural network [2]

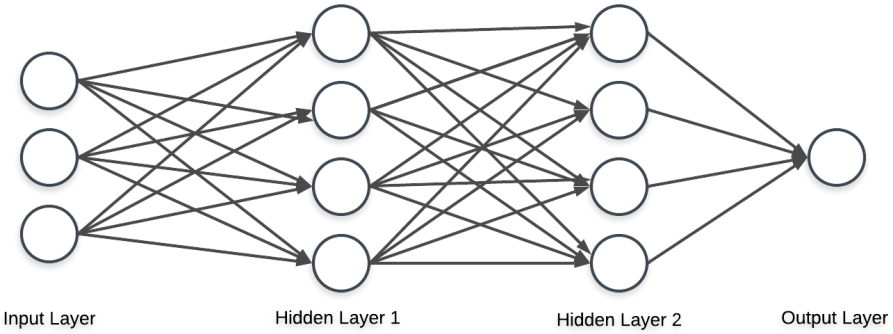


Figure 2.10: Multi-layer neural network [43]

- **Activation Functions**

  Activation functions introduce non-linearity in the network's computations, which is crucial for modelling complex relationships. Without these functions, the output would be a linear combination of input values, severely limiting the network's ability to learn intricate patterns. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh, each contributing differently to the model's performance.

- **Weights**

  Weights assign importance to features that significantly contribute to the learning process. They do this by scalar multiplication between the input values and the weight matrix. For instance, in sentiment analysis, a negative word impacts the decision more than neutral words by altering the weight applied. This mechanism allows the model to prioritize relevant features during learning.

- **Bias**

  Bias helps to shift the value produced by the activation function, similar to the role of a constant in a linear function. This adjustment allows the network to better fit the data by modifying the output alongside the weighted sum of inputs. Bias thus enhances the model's flexibility and accuracy.

- **Learning Rule**

  The learning rule is a set of guidelines that dictates how the weights and biases are adjusted during the training process. It is essential for the network to learn from data effectively. This rule typically involves algorithms such as gradient descent, which minimizes the error by updating the weights and biases in the direction that reduces the loss function. The learning rule ensures that the model converges towards optimal performance.

- **Input Layer**

  The input layer consists of the set of features fed into the model for the learning process. For example, in object detection, the input can be an array of pixel values corresponding to an image. This layer acts as the interface for data from external sources, such as CSV files or web services, to be loaded into the model. It is the only visible layer in the neural network architecture that transmits complete information from the outside world without any computation.

- **Hidden Layers**

  Hidden layers are the layers found between the input layer and the output layer in a neural network. They are responsible for processing data and extracting important features. They are crucial for deep learning, enabling the model to learn hierarchical feature representations. Multiple interconnected hidden layers can exist, each responsible for identifying different features. For instance, initial hidden layers might detect edges and shapes in image processing, while later layers identify complex objects such as cars, buildings, or people.

- **Output Layer**

  The output layer receives input from the preceding hidden layers and makes the final prediction based on the model's learned features. This layer is critical for obtaining the final result of the model's decision-making process. It synthesizes the information processed by the hidden layers to produce a conclusive output.

2.4.2 Activation Functions

Activation functions are critical components of neural networks, as they introduce non-linearity into the model, which is essential for the network's ability to capture and learn intricate patterns within the data. The choice of activation function can significantly influence both the learning process and the overall performance of the network.

In our implementation, we experimented with three key activation functions: sigmoid, tanh, and ReLU. This approach allowed us to evaluate their respective performances and select the most suitable one based on our specific requirements. However, for classification of the activation functions, they are divided into two types [19]:

1. **Linear Activation Function**

   The linear activation function, also known as the identity function, is a fundamental component of neural networks. It is a linear function with a range of (-∞, ∞) and is directly proportional to the input, non-linear, and differentiable, with a constant derivative of 1, [19].

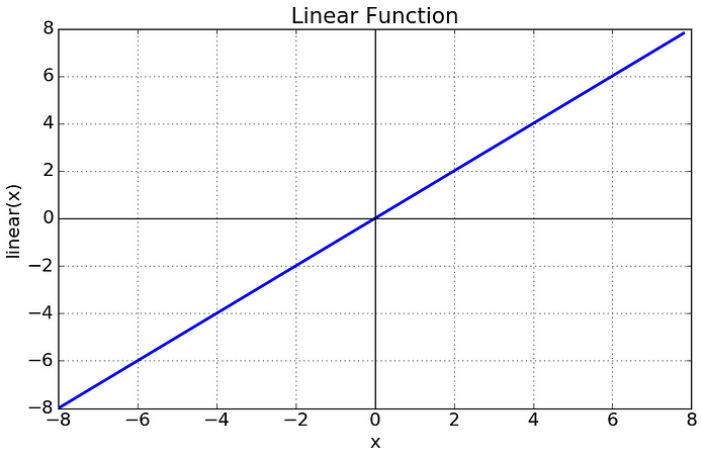


Figure 2.11: Linear Activation Function [19]

2. **Non-Linear Activation Functions**

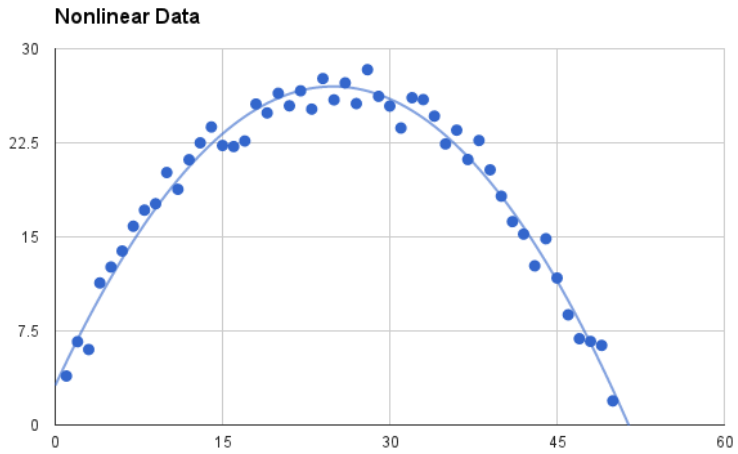


Figure 2.12: Non-Linear Activation Function [19]

A non-linear activation function (Fig. 2.12) is a crucial component in neural networks, introducing non-linearity into the network's decision-making process. Here are some commonly used non-linear activation functions:

- **Sigmoid Function (Logistic)**
  The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

  The sigmoid activation function curve looks like a S-shape. It accepts any real number as input and produces output values ranging from 0 to 1. As the input becomes larger (more positive), the output value approaches 1. Conversely, as the input becomes smaller (more negative), the output value nears 0, as illustrated in Fig 2.13, [19].

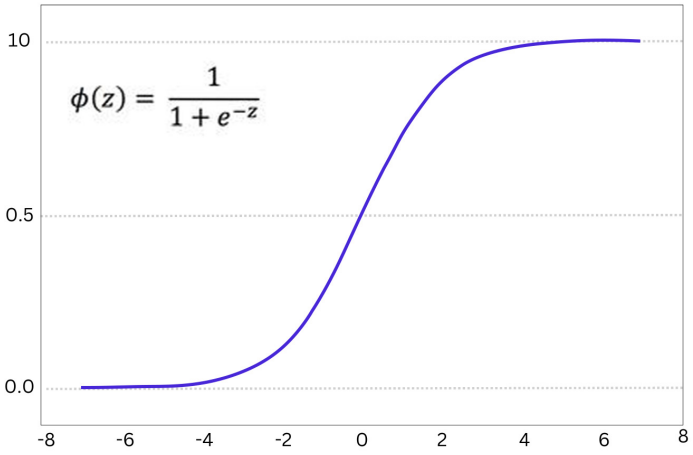


Figure 2.13: Sigmoid/Logistic Function [19]

- **Hyperbolic Tangent Function (Tanh)**
  The hyperbolic tangent activation function is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.5}$$

The Tanh function is similar to the sigmoid activation function, sharing the same S-shaped curve. However, it differs in its output range, which spans from -1 to 1. In the Tanh function, as the input becomes larger (more positive), the output value approaches 1 and, as the input decreases (becomes more negative), the output value approaches -1 [19], as shown in Fig 2.14

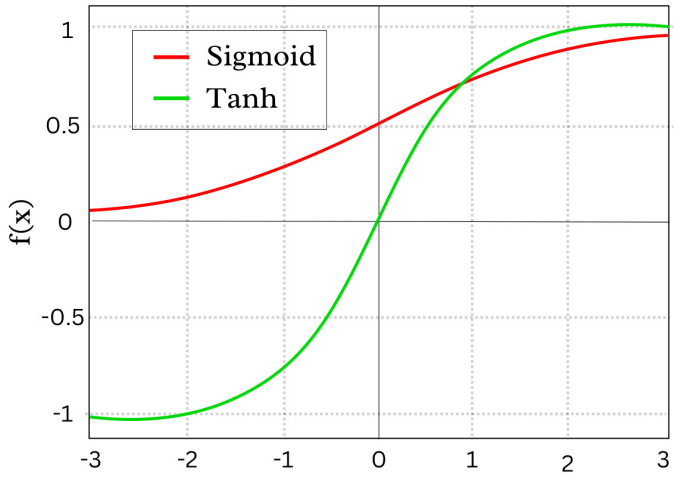


Figure 2.14: Tanh Function [19]

- **Rectified Linear Unit (ReLU)**
  ReLU stands for Rectified Linear Unit. It is defined as:

$$\mathrm{R}(z) = \max(0, z) \tag{2.6}$$

While it appears linear, ReLU includes a derivative function that enables efficient backpropagation. The key feature of the ReLU function is its selective activation of neurons. As compared to the

sigmoid function, neurons remain inactive if the output of the linear transformation is less than 0. See Fig 2.15.

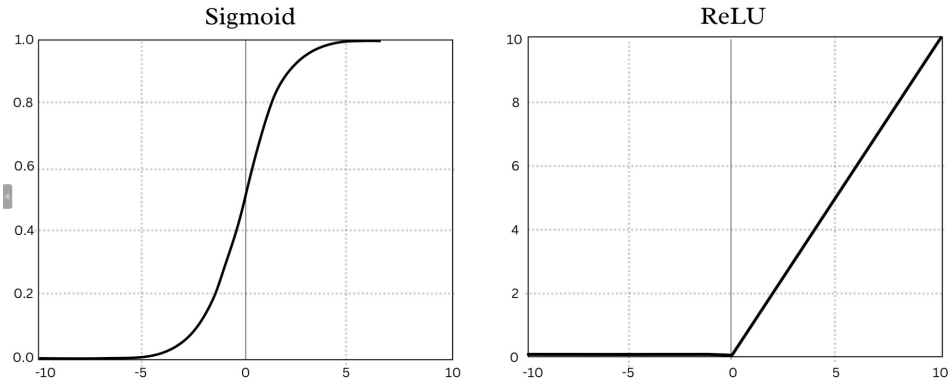


Figure 2.15: ReLU Function [19]

It encounters the Dying ReLU issue, where the gradient becomes zero for negative input values. As a result, during backpropagation, the weights and biases of certain neurons fail to be updated, leading to inactive neurons that never activate [19].

2.4.3 Types of Neural Networks

Neural networks can be categorized based on various criteria, including their structure, data flow, neuron types and density, layer configuration and depth, activation functions, and other characteristics. These classifications encompass existing neural network models and those under development [43].

- **Single Layer Perceptron**

  This is the simplest type of feed-forward neural network, where inputs are directly transmitted to the outputs via a series of weights, as shown in Fig 2.16. At each node, the sum of the weighted inputs is calculated and compared to a threshold value. If the computed value exceeds the threshold, the neuron activates and is assigned a value of 1; otherwise, it deactivates and is assigned a value of -1. It has been demonstrated that networks of parallel threshold units can approximate any continuous function over a finite range of real numbers within the interval [-1, 1]. Single-layer perceptron neural networks are typically used to solve simple problems, and their low computational cost allows for quick results [2, 43].

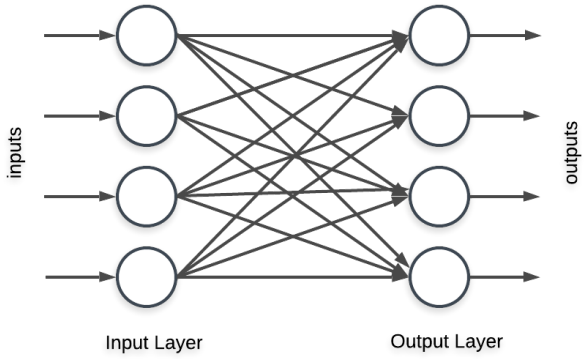


Figure 2.16: Single Layered Perceptron [2]

- **Multi-Layer Perceptron**

  This model can have zero or more hidden layers between the input and output layers, as illustrated in Fig 2.10, which shows two hidden layers. Unlike single-layer perceptron models, multi-layer perceptron

models can solve non-linear problems, including non-linearly separable issues like XOR. The number of neurons in the input layer corresponds to the number of features in the pattern recognition problem. In contrast, the number of neurons in the output layer corresponds to the number of classes. Training a multi-layer perceptron model can be done using the Backpropagation Algorithm. Multi-layer perceptron models can handle complex problems. However, they require significant computational resources due to the need for many iterations during the learning process, making them less suitable for real-time learning [2, 43].

- **Recurrent Neural Network (RNN)**

  RNNs [43], address the ability to build upon prior knowledge, by incorporating loops in their architecture, which allows data to persist within the network. The connections between the nodes in RNNs form a directed graph along a sequence. Unlike feed-forward neural networks, RNNs utilize their internal state or memory to process sequences of inputs. An RNN comprises an input, hidden, and output layer, as illustrated in Fig 2.17. The input layer integrates h_{t-1} and x_t, where h_{t-1} is the vector input from previous information (from time 0 to $t-1$) and x_t is the corresponding input vector at time t. The hidden layer processes these inputs to produce a history vector h_t, which is then used to predict future outputs. For an input sequence x = $(x_1, ..., x_T)$, the RNN calculates the hidden layer vector sequence h = $(h_1, ..., h_T)$ and the output layer vector sequence y = $(y_1, ..., y_T)$ using the equations:

$$h_t = \omega(W_{ih}x_t + W_{hh}h_{t-1} + b_h) \tag{2.7}$$

$$y_t = W_{ho}h_t + b_o \tag{2.8}$$

  Where W expresses the weight matrices b expresses the bias vector and ω is the hidden layer activation function.

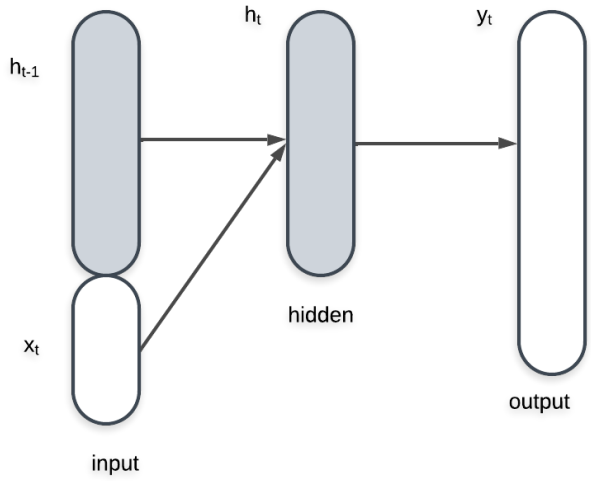


Figure 2.17: Recurrent Neural Network [43]

Long Short-Term Memory (LSTM) networks, a variant of Recurrent Neural Networks (RNNs), incorporate specialized units in addition to standard units. These specialized units, known as LSTM cells, contain a 'memory cell' capable of maintaining information over extended periods. The regulation of information flow within these cells is managed by a set of gates: the input gate, the output gate, and the forget gate. The input gate determines the extent of information from the previous sample to be retained in memory; the output gate controls the amount of information transmitted to the next layer; and the forget gate manages the rate at which stored information is discarded. This architectural design enables LSTM networks to effectively learn and retain long-term dependencies [43].

- **Convolutional Neural Network (CNN)**

  Convolutional Neural Networks (CNNs) are similar to traditional Artificial Neural Network (ANN)s in that they consist of neurons that optimize themselves through learning. However, a key distinction between CNNs and ANNs is that CNNs are primarily utilized for pattern recognition within images. This allows CNNs to encode image-specific features into their architecture, making the model particularly well-suited

for tasks related to image processing. CNNs fundamentally rely on the assumption that the input data will consist of images. Their architecture is designed to optimally process this specific type of data. The architecture of CNNs mainly comprises three types of layers: convolutional layers, pooling layers, and fully connected layers [2, 43]. The simple architecture of a CNN is depicted in Fig 2.18.

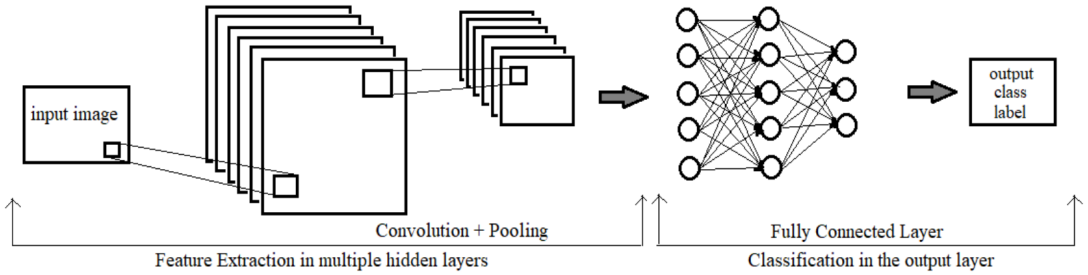


Figure 2.18: CNN Architecture [43]

The convolutional layer performs convolution operations on the input data, utilizing filters to extract local features such as edges, textures, and patterns. Following this, the pooling layer executes down-sampling to reduce the spatial dimensions of the feature maps, which not only decreases the computational load but also helps mitigate overfitting. At the end of the network, the fully connected layer consolidates the features extracted by the convolutional and pooling layers to generate the final classification or prediction. The architecture and functioning of the CNN are discussed in greater detail in the upcoming Sec. 2.5.

2.5 Convolution Neural Networks

Building on the foundational concepts of neural networks, we now focus on CNNs. CNNs are a specialized type of neural network particularly effective for processing grid-like data, such as images. This section will discuss the structure, functionality, and applications of CNNs, emphasizing their importance in computer vision and other fields.

2.5.1 Structure of a Convolutional Network

In CNNs, the states in each layer are organized in a spatial grid structure, as shown in Fig. 2.18. These spatial relationships are preserved from one layer to the next, as each feature value is derived from a small local region in the previous layer. Maintaining these spatial relationships among grid cells is crucial because the convolution operation and the subsequent layer transformations depend on them. Each layer in a CNN forms a 3-dimensional grid structure, comprising $height$, $width$, and $depth$. In the context of a single layer in CNN, depth refers to the number of channels in that layer, such as the primary colour channels (blue, green, and red) in an input image. It is also used to describe the number of layers in a network. A CNN operates similarly to a traditional feed-forward neural network but with spatially organized layers and sparse, well-designed connections between them. The three common types of layers in a CNN are convolutional, pooling, and Rectified Linear Unit (ReLU). The ReLU activation functions (Fig. 2.15), are the same as those in traditional neural networks. Additionally, the final layers are often fully connected and map to a set of output nodes in an application-specific manner [2, 29].

2.5.1.1 Convolutional Layer

The convolutional layer is integral to the functionality of CNNs, focusing on the use of learnable kernels. These kernels, although small in spatial dimensions, extend across the entire depth of the input. When data is processed through a convolutional layer, each filter convolves across the spatial dimensions of the input, generating 2D activation maps. These maps can be visualized and are calculated by performing scalar products for each value in the kernel, as depicted in Fig 2.19. The network learns kernels that activate when they detect

specific features at particular spatial positions in the input, resulting in $activations$. Each kernel produces a corresponding activation map, which is stacked along the depth dimension to form the convolutional layer's output volume [29].

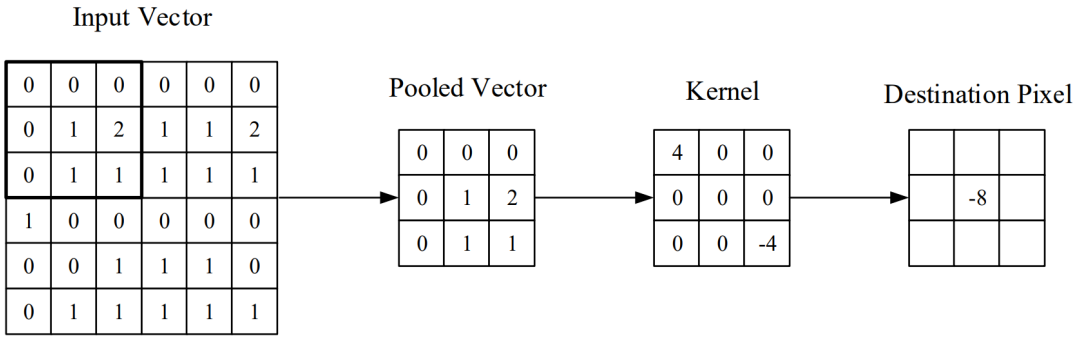


Figure 2.19: A visual representation of a convolutional layer: the central element of the kernel is positioned over the input vector, then computed and replaced with a weighted sum of itself and the surrounding pixels [29]

Training ANNs on image data typically results in models too large to train efficiently, due to the fully connected nature of standard ANN neurons. To address this, neurons in a convolutional layer connect only to a small region of the input volume, referred to as the receptive field size. The connectivity depth is generally equal to the depth of the input. For example, an input image of size 64 × 64 × 3 (RGB image) with a receptive field size of 6 × 6 would have 108 weights per neuron in the convolutional layer, significantly fewer than in a standard ANN neuron.

Convolution layers reduce model complexity by optimizing output through hyper-parameters such as $depth$, $stride$, and $zero - padding$. The depth of the output volume can be set by the number of neurons within the layer. The stride parameter determines the depth around the spatial dimensions of the input, affecting the level of overlap in the receptive field and the spatial dimensions of the output. Zero-padding involves padding the input border to control output dimensions. The spatial dimensions of the output are altered using these techniques, calculated using the formula:

$$\frac{(V - R) + 2Z}{S} + 1 \tag{2.9}$$

where V is the input volume size, R is the receptive field size, Z is the amount of zero padding, and S is the stride. The result must be a whole number to ensure neurons fit neatly across the input [2, 29].

2.5.1.2 Pooling Layer

Pooling is a critical component of CNN architectures, designed to streamline feature maps by condensing their information into a smaller, more manageable form. This process preserves key features while discarding unnecessary details, enhancing the overall utility and efficiency of the representation. Additionally, pooling introduces spatial invariance, making the network less sensitive to the exact positioning of features within the input. By reducing the number of connections between layers, pooling not only lightens the computational load on subsequent layers but also maintains essential information. In essence, it down-samples the output from the previous layer, producing lower-resolution feature maps that still capture the most important details [13]. The pooling layer applies various mechanisms such as average-pooling, max-pooling, mean-pooling, and min-pooling to the feature map.

One commonly used method is max-pooling, which typically employs a 2 × 2 kernel with a stride of 2. This configuration reduces the activation map to 25% of its original size while preserving the depth of the volume, as illustrated in Fig. 2.20. However, the potentially destructive nature of pooling can influence model performance. Two prevalent approaches for max-pooling are often utilized: the first sets both the stride and filter size to 2 × 2, covering the entire spatial dimensionality of the input. The second employs overlapping pooling with a stride of 2 and a kernel size of 3, though it is important to note that larger kernel sizes may negatively impact model performance [2, 29].

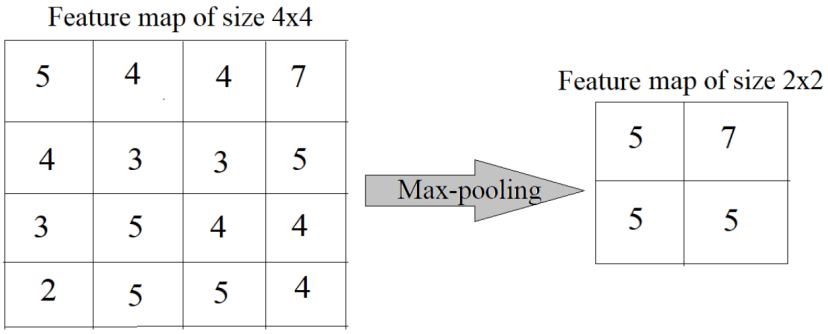


Figure 2.20: Max Pooling[29]

2.5.1.3 Fully-connected Layer

In a convolutional neural network, Fig 2.18, each feature in the final spatial layer connects to every hidden state in the first fully connected layer, functioning like a traditional feed-forward network. Typically, multiple fully connected layers are used to enhance computational power towards the end. These layers are structured similarly to traditional feed-forward networks, with dense connections resulting in the majority of parameters residing in the fully connected layers. For instance, if two fully connected layers each have 4096 hidden units, the connections between them exceed 16 million weights. Similarly, the connections from the last spatial layer to the first fully connected layer contain a large number of parameters. Although convolutional layers have more activations and therefore a larger memory footprint, fully connected layers usually have a greater number of connections and parameters. This is because activations, multiplied by mini-batch size, significantly contribute to the memory footprint, especially when tracking variables during forward and backward passes of back-propagation. The nature of the fully connected layer can vary depending on the application, such as classification or segmentation. In classification, the output layer is fully connected to every neuron in the penultimate layer with associated weights and may use logistic, softmax, or linear activation functions. In segmentation tasks, rather than a fully connected output layer, a series of deconvolutional or upsampling layers are often used to reconstruct the spatial dimensions of the input image [2, 29].

2.5.2 Data pre-processing

Data pre-processing is a critical phase in preparing datasets for training machine learning models, particularly neural networks, as it enhances the model's generalization capabilities and ensures robust performance across various tasks. The first step in this process is data cleaning, which involves identifying and correcting issues within the dataset to ensure its quality and reliability for subsequent analysis. This includes handling missing values, which may occur due to data entry errors or incomplete records, through techniques such as imputation, algorithms capable of handling missing data, or excluding records with missing values [18]. Additionally, removing duplicates ensures that each data point is unique, avoiding redundancy and ensuring diverse examples for model training. Outlier detection and removal, using methods such as statistical tests, visualization, or domain-specific knowledge, is also crucial to prevent outliers from adversely affecting the analysis [32].

Another vital pre-processing step is normalization, which involves scaling input data to a specific range, typically between 0 and 1 or -1 and 1. This process enhances the convergence speed and stability of training by ensuring that all features contribute equally, thus preventing larger-scale features from disproportionately influencing the learning process. It also helps mitigate issues like exploding and vanishing gradients, which can hinder the training of deep neural networks [16]. Common techniques for normalization include min-max scaling and z-score normalization, both of which have been shown to improve the efficiency and accuracy of neural network models [6].

Data splitting is another crucial step in preparing data for training machine learning models. This involves partitioning the dataset into distinct subsets to evaluate and validate model performance effectively. The primary splits are the train-test split and cross-validation. The train-test split divides the dataset into two parts: the training set, used to train the model, and the test set, reserved for evaluating its performance on unseen data, thus helping to assess the model's generalization to new data [9]. Cross-validation involves dividing the

dataset into multiple folds, where the model is trained and validated on different subsets iteratively, providing a more robust evaluation by averaging performance metrics across different folds, thereby reducing the risk of overfitting [8].

Data augmentation is another technique used to artificially increase the size and diversity of a dataset by generating modified versions of existing data samples. This is particularly useful in machine learning and neural network training, where large, diverse datasets are often required to improve model performance and generalization. For image data, common augmentation methods include transformations such as rotation, scaling, flipping, and colour adjustments, which help the model learn to recognize patterns under various conditions [45]. By expanding the dataset with augmented examples, the model becomes more robust and less likely to overfit, ultimately leading to better performance on unseen data [34].

2.6 U-Net Architecture

In this section, we delve into the U-Net architecture, which serves as the foundation of our convolutional network. U-Net is a specialized CNN, prepared specifically for image segmentation tasks, where the goal is to classify each pixel in an image. Its design follows an encoder-decoder structure: the encoder captures and extracts hierarchical features through convolution and pooling operations, while the decoder works to reconstruct these features into spatial details using up-sampling. One of the standout features of U-Net is the incorporation of skip connections. These connections link corresponding layers in the encoder and decoder, effectively transferring high-resolution spatial information to the up-sampling path.

Figure 2.21 illustrates an example of a U-Net architecture tailored for a biomedical image segmentation task [35]. This architecture has been modified to perform effectively with a significantly smaller number of training images while still producing accurately segmented outputs. Each blue box denotes a multi-channel feature map, with the number of channels specified at the top of the box. The x-y dimensions are shown in the lower-left corner of the box. White boxes denote feature maps that have been copied, and the arrows depict the various operations occurring throughout the network. Due to its U-shaped architecture, the network is capable of capturing both local features and global context, leading to precise segmentation results.

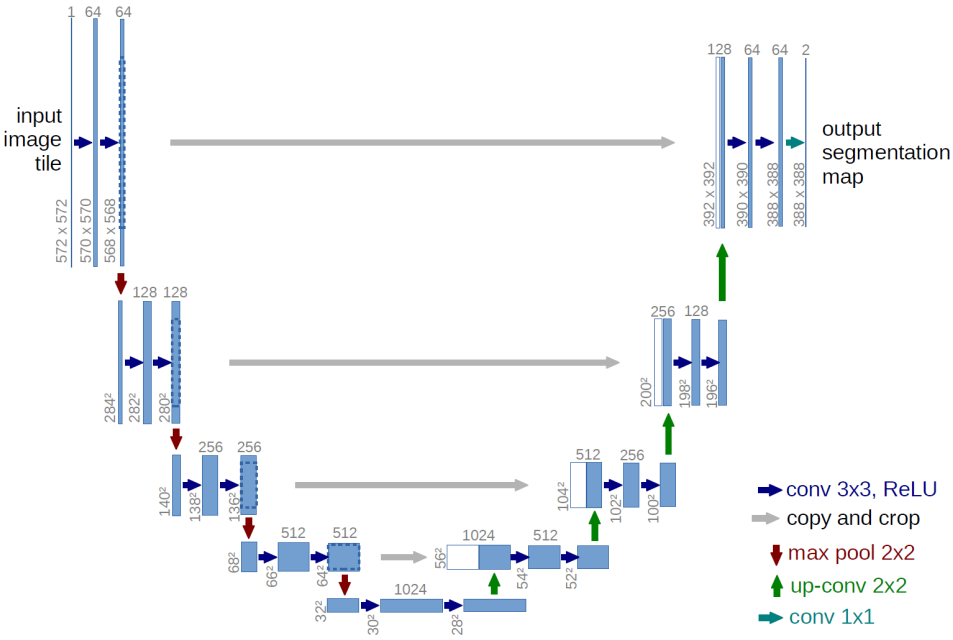


Figure 2.21: Representation of the U-Net Architecture [35]

The U-Net architecture consists of several key components that contribute to its distinctive U-shaped design, [35, 40]:

- **Contracting Path (Encoding Path):** The contracting path of the U-Net architecture comprises a series of convolutional layers, each followed by a ReLU activation and max pooling operations. This path effectively captures high-resolution, whilst progressively lowering the spatial dimensions of the image. As the feature maps pass through these layers, they become increasingly abstract, allowing the network to learn essential patterns and details crucial for segmentation. The combination of convolutional operations and pooling helps to highlight important features such as edges and textures, forming a rich representation of the input image that will later be utilized in the expansive path for precise segmentation.
- **Expanding Path (Decoding Path):** While the expanding path uses transposed convolutions or deconvolutions of the feature maps for the up-sampling layer. These links enable the network to collect local as well as global data. By integrating feature maps from previous layers through skip connections, the network retains crucial spatial details, significantly enhancing segmentation accuracy. This helps the model better define object boundaries and improves its ability to identify and segment different structures in the image. The combination of high-level features from the decoder and precise spatial information from the encoder results in more refined outputs, making U-Net particularly effective for detailed segmentation tasks, such as in medical imaging, where accuracy is vital.
- **Skip Connections:** These connections are used in the architecture to keep important details that can get lost during the down-sampling of an image. They link feature maps from the encoder to those in the decoder, ensuring that crucial spatial information is retained. By matching corresponding layers, skip connections allow the model to combine low-level features with higher-level insights. This integration enhances the network's ability to accurately reconstruct and segment images, improving segmentation accuracy and helping to define object boundaries more clearly. This makes the U-Net especially effective for tasks where precision is essential.
- **Concatenation:** Concatenation is a crucial operation, particularly in the context of skip connections. It involves combining feature maps from the encoder and decoder paths. When the feature maps from a layer in the encoder are concatenated with those from the corresponding layer in the decoder, it enables the model to retain important spatial details while leveraging the high-level features learned in the decoder. this creates a better representation of the image.
- **Fully Convolution layers (FCLs):** The FCLs work by applying convolutional operations to the entire input image, effectively allowing the network to learn spatial hierarchies without losing the spatial structure of the input. Unlike traditional neural networks that rely on fully connected layers at the end, U-Net uses only convolutional layers throughout its architecture. This design allows the network to accept input images of varying sizes and produce output segmentation maps of the same dimensions.

The U-Net architecture excels in image segmentation by capturing both local and global features through its core components. Its ability to train effectively on limited data, while preserving essential spatial details and combining them with high-level abstractions, makes it a foundational element for this research work.

2.7 Transfer Learning with ResNet

With transfer learning, a model trained for one task is repurposed as the foundation for a new model tackling a different but related task. Instead of training a model from scratch, transfer learning uses the knowledge gained from a pre-trained model, typically trained on a large dataset, to improve learning efficiency and performance on a new task with less data or computational resources. A variety of CNN architectures have been developed such as the AlexNet, VGGNet, ResNet, etc to facilitate this process [14]. These architectures have been trained on extensive datasets, allowing them to capture rich and meaningful feature representations. In this study, we specifically examine the use of ResNet as the backbone of our network for the motion segmentation task, utilizing its powerful feature extraction capabilities to improve the overall performance of the network.

The Residual Network or $ResNet$ [14], is a deep learning architecture designed to enable its weight layers to learn residual functions based on the inputs from the preceding layers. A defining feature of ResNet is its incorporation of residual connections, which facilitate the smoother flow of gradients during backpropagation. This characteristic is crucial for training networks that contain hundreds or even thousands of layers.

A residual connection specifically denotes the architectural pattern characterized by the mathematical transformation $x \mapsto f(x) + x$, where x represents the input layer and f signifies an arbitrary neural network module, as depicted in Fig. 2.22 on the left. This transformation allows the output of the function $f(x)$ to be combined with

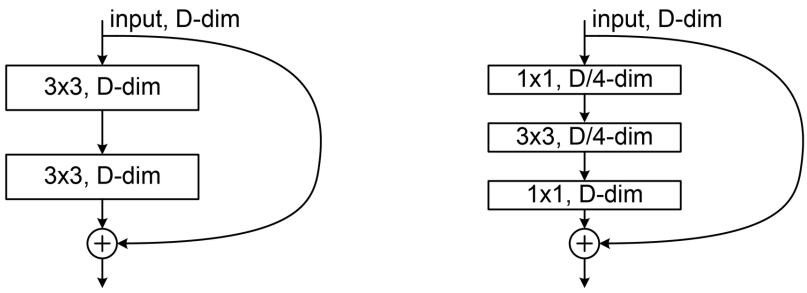


Figure 2.22: Types of Residual block [14, 15]

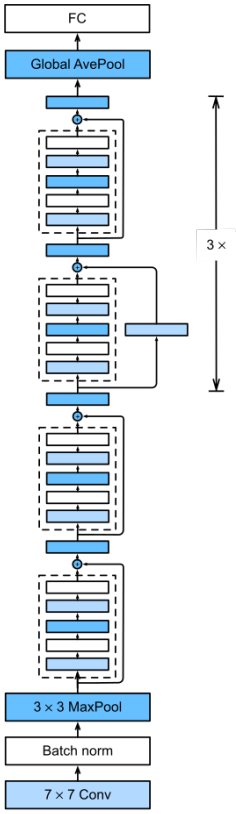


Figure 2.23: Overview of the ResNet architecture [15]

the original input x by a $+$ operation, which signifies a skip connection which performs an identity mapping. Another use case would be in the form of a bottleneck block, as illustrated in Fig. 2.22 on the right, where a 1x1 convolution block is used for dimension reduction, followed by a 3x3 block, and at the end there is another 1x1 convolution for dimension restoration. These residual connections are the fundamental building blocks of the ResNet architecture. By stacking multiple residual blocks, ResNet allows for the construction of deep neural networks while maintaining efficient gradient flow through the skip connections. Fig. 2.23 shows a high-level overview of the ResNet architecture. It starts with a 7x7 convolutional layer followed by batch normalization, a 3x3 max-pooling layer, and then several residual blocks, which include skip connections that help the network learn more efficiently by bypassing layers. After the residual blocks, global average pooling reduces the data's dimensionality before passing it to a fully connected layer for the final prediction. Each residual block contains a series of transformations that learn the residual mapping, and the skip connections ensure that the original input is passed through directly, facilitating easier optimization. This modular design allows ResNet architectures to scale up to hundreds or even thousands of layers. The skip connections allow the network to preserve essential information across layers, making it possible for very deep networks to retain accuracy and generalize effectively [14].

3 Literature Review

Motion segmentation has long been a central problem in computer vision, especially in scenarios where distinguishing between independently moving objects and background motion is essential. Traditional frame-based cameras were initially the dominant tools for addressing this task. These cameras, however, faced persistent challenges such as motion blur and degraded performance in dynamic or high-speed scenes. In response, event cameras like the DVS have emerged as alternatives, offering asynchronous data capture and high temporal resolution. Event cameras record changes in brightness at each pixel in real time, avoiding much of the redundancy and lag associated with frame-based cameras. As a result, they have paved the way for novel approaches to motion segmentation, spanning from classical techniques to modern machine learning-based methods. This review will explore these developments, analyzing the progression from parameter-driven methods to more flexible, data-driven approaches, while highlighting the key contributions and limitations in existing research.

In the early stages of the research, traditional methods focused heavily on frame-based approaches. One widely studied method was optical flow, which tracks pixel intensity changes between consecutive frames to estimate motion. The foundational work in the paper [17] introduced an algorithm for calculating optical flow using image gradients. This method became a cornerstone of early motion segmentation research, yet it struggled with complex motion dynamics, particularly in high-speed scenarios. Motion blur and the relatively low temporal resolution of frame-based cameras posed significant limitations in dynamic environments, diminishing the effectiveness of optical flow techniques. Another popular approach, background subtraction, aimed to model static scenes and identify moving objects based on deviations from this model. The paper [37] extended this concept with the introduction of Gaussian Mixture Models for background modelling, which proved effective for segmenting objects in relatively controlled environments. However, like optical flow, background subtraction faced substantial hurdles when applied to more dynamic scenes where rapid object motion or frequent lighting changes compromised its ability to distinguish between moving objects and background elements.

Event cameras, such as the DVS, tackle many challenges faced by traditional cameras, particularly in scenarios requiring high temporal precision. Unlike frame-based cameras, event cameras capture data asynchronously, recording only pixel-level brightness changes, which significantly reduces motion blur and enables accurate tracking of fast-moving objects. One seminal study [48] introduced a spatio-temporal graph-cut technique for motion segmentation using event data, modelling the task as an energy minimization problem that groups events into clusters corresponding to different motion models, thereby overcoming the limitations of traditional methods. Building on this foundation, the research on unsupervised deep event stereo [42] demonstrates innovative depth estimation techniques that utilize event data to reconstruct, which is crucial in providing the spatial information of the object, that further distinguishes it from other objects or background.

In response to the limitations of predefined models, several machine-learning techniques for motion segmentation were explored. The transition from classical to data-driven methods marked a significant shift in the field, as machine learning approaches offered greater flexibility and adaptability to dynamic environments. A noteworthy study in the paper [31], introduced an unsupervised learning framework for event-based motion segmentation. This method utilized clustering techniques to group events based on motion characteristics without relying on prior knowledge or predefined models. While this represented a crucial step forward in reducing the need for manual parameter tuning, unsupervised methods still struggled with handling complex environments, particularly those with multiple independently moving objects or noisy data. In such cases, segmentation errors were more likely to occur, indicating that further refinement was needed. A more recent study [1] proposed using Graph Neural Networks (GNN) for unsupervised motion segmentation. This approach

enhances the clustering process by leveraging deep features from vision transformers for graph construction, thereby improving the segmentation's accuracy, especially in dynamic environments. Their method eliminates the need for extensive post-processing by applying a two-stage clustering approach, which improves both the granularity of foreground segmentation and overall system robustness.

Integrating deep learning models, particularly CNNs, has significantly advanced the analysis of event data. CNNs are well-suited for processing spatial data, making them ideal for the rich information captured by event cameras. The study in the paper [22], pioneered this approach by transforming event data into voxel grids, enabling the application of CNNs originally designed for conventional image data. Their model demonstrated marked performance improvements compared to traditional methods. This research highlighted the potential of CNNs to effectively handle the complex, asynchronous data produced by event cameras, establishing a solid foundation for future innovations in areas such as autonomous driving technologies.

Building on the success of CNNs, deeper architectures like ResNet have significantly improved motion segmentation capabilities. By introducing residual connections, ResNet allows for the training of much deeper networks while effectively addressing the vanishing gradient problem. The work done in [14] utilizes ResNet to create an optimized model specifically designed for event camera data. They highlighted the importance of meticulous hyperparameter tuning—such as selecting the right temporal resolutions and transforming event data into frame-like representations—to enhance both segmentation accuracy and computational efficiency. This study emphasizes the vital role that deep learning models play in tackling the complexities of real-world motion segmentation, especially when dealing with high-frequency data generated by event cameras.

A major challenge in implementing machine learning techniques is the intricate process of hyperparameter tuning, which requires fine-tuning various parameters that can greatly influence the model's effectiveness. These adjustments play a crucial role in shaping the model's performance and overall results. The paper [46] explores the art of fine-tuning machine learning models. They delve into techniques that can boost both the speed and accuracy of these models. The authors offer a detailed guide to various optimization strategies, covering everything from the theory to the practical implementation. They also introduce a range of tools and libraries that can simplify the process of optimizing hyperparameters. While the paper highlights the challenges involved in hyperparameter optimization, it also provides valuable insights and practical examples. Through experiments on real-world datasets, the authors demonstrate different optimization techniques and their effectiveness.

In conclusion, our research demonstrates that spatio-temporal graph-cut techniques combined with neural networks are effective for motion segmentation and depth estimation in dynamic environments. However, current methods face challenges in managing noise and uncertainties, as well as in maintaining robust generalization across varying scenarios, particularly when utilizing event cameras. These limitations highlight opportunities for further exploration, especially in enhancing noise resilience and improving model adaptability to diverse settings. In this study, we first implemented a graph-cut-based approach [48] for motion segmentation using event camera data to assess its effectiveness. Additionally, we developed a learning pipeline comprising two neural networks, inspired by the works of [47, 25]. Our primary focus is to build upon these established approaches in the field of event-based segmentation, aiming to refine their performance and applicability.

4 Methods

This chapter describes the implementation of a motion segmentation method based on an energy minimization framework [48], which fits multiple motion models to a spatio-temporal graph constructed from event camera data, to achieve state-of-the-art results across various datasets with different motion patterns and numbers of moving objects. Subsequently, a learning pipeline is implemented [25], which estimates objects' depth and motion flow using neural networks, providing a more robust approach to handling motion segmentation in dynamic environments.

4.1 Motion Segmentation using Energy Minimization

This section describes a classical method for motion segmentation using event-based cameras. The key idea is to approach the problem as an energy minimization task, where the energy function is designed to balance three factors: how well the data fits the motion models, the spatial consistency of the segmentation, and the simplicity of the overall model.

4.1.1 Problem Formulation

The method aims to organize events into clusters representing different Independent Moving Object (IMO)s within the scene. After clustering, the events are warped according to their estimated motion models, producing an Images of Warped Event (IWE) with enhanced contrast, as detailed in [48]. Fig 4.1 highlights the key stages of the method, from initial data representation to motion compensated events.

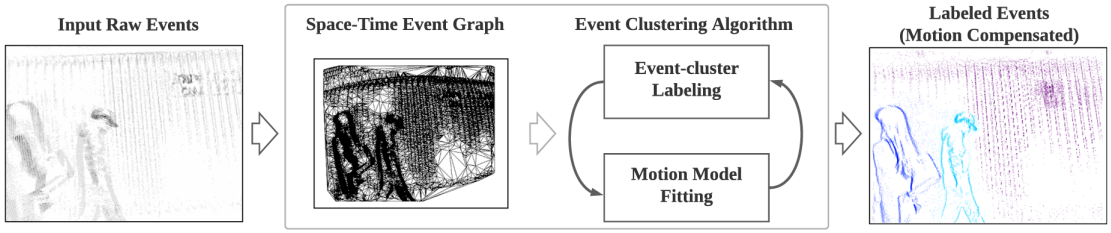
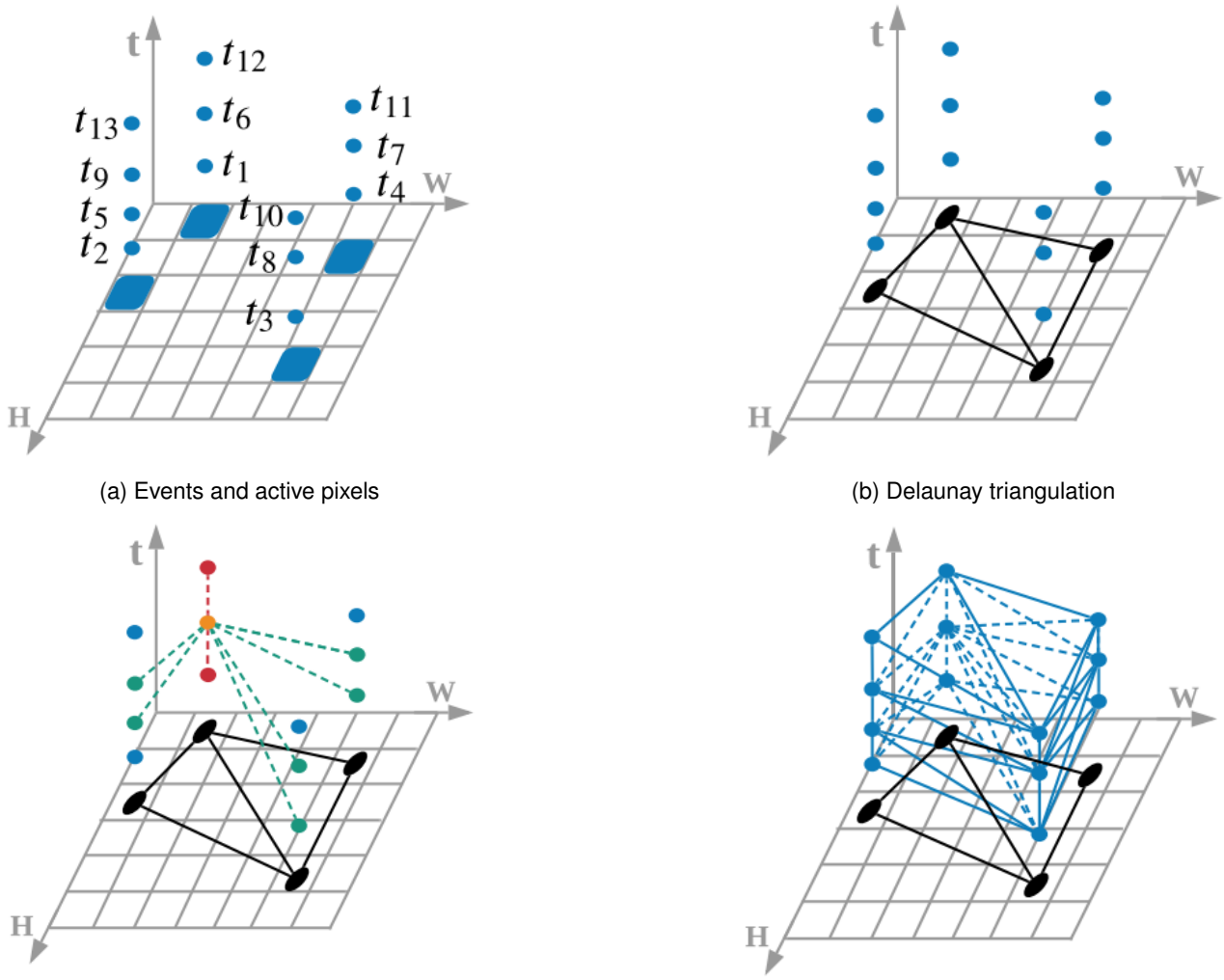


Figure 4.1: Flowchart of the motion segmentation process using event-based cameras [48]

4.1.2 Space-Time Graph Model Construction

The initial step is to transform the raw, asynchronous event data into a structured representation that captures the inherent spatio-temporal relationships between events. The process begins with a 3D space-time volume V of size W x H x δt (referring to the dimension of the image and the time span) where events are represented as points. It then creates a 2D binary image where a pixel is '1' if it has at least one event in the corresponding

space-time volume, and '0' otherwise. This image shows which pixels were 'active' during the evaluated period, as shown in Fig 4.2a.



(a) Events and active pixels

(b) Delaunay triangulation

(c) Connecting principle

(d) Spatio-temporal graph

Figure 4.2: Construction of the spatio-temporal graph [48]

A Delaunay triangulation is performed on the 'active' pixels in the binary image. This results in a 2D mesh, Fig 4.2b, where the active pixels are the vertices, and edges connect nearby pixels given in Fig 4.2c. The final 3D spatio-temporal graph is built by connecting each event to its neighbours in both space and time, where each event has 2 + 2N neighbours (N denotes number of edges to the event's pixel location) in the resulting graph (Fig 4.2d), [48].

4.1.3 Multi-Model Fitting

The core multi-model fitting approach is applied to the event data. This step is sub-divided into two tasks: each event is assigned to one of the motion models (labelling), in a way that maximizes the consistency between the event's motion and the assigned model's motion, and the other step refines the parameters of each motion model to better fit the events assigned to it, based on the motion compensation technique to align the events along their motion trajectories (parameter estimation), [48].

1. **Labelling:** This step focuses on classifying each event by assigning it to the most suitable motion model or cluster. The assignment is driven by the objective of maximizing the consistency between the motion

characteristics of an event and the motion represented by the assigned model. The underlying principle is to group events exhibiting similar motion patterns together, [48].

2. **Parameter Estimation:** Once the events have been assigned to their respective clusters, this step refines the parameters of each motion model to ensure they optimally represent the motion of the associated events. A motion compensation technique relies on warping events according to a motion model and aggregating them into an IWE. The sharpness or contrast of the IWE serves as an indicator of how well the events align with the model. The method seeks to find the motion model parameters that maximize the IWE's contrast, thereby achieving a better fit between the model and the events, [48].

4.1.4 Energy minimization

The process then transitions to energy minimization, which is the mathematical framework that leads the solution to the multi-model fitting problem. The energy function serves as a quantitative measure of how well a particular combination of motion models and event assignments, explains the observed data. The lower the energy value, the better the fit, [48]. The three key aspects which combine to form the energy minimization process are:

1. **Data Fidelity (λ_D):** The algorithm first computes the negative IWE for each event and its assigned motion model. This value is used to quantify the alignment between the observed event data and the model prediction. It repeatedly adjusts the event assignments to different motion models, driving the segmentation toward a configuration where the motion patterns are best represented by the models.
2. **Spatial Coherence (λ_P):** Next, the algorithm enforces spatial coherence (λ_P) by constructing a spatio-temporal graph of the events. This step encourages smoothness in the labelling by minimizing the number of label changes between adjacent events, ensuring a consistent segmentation.
3. **Minimum Description Length (MDL)(λ_M):** The algorithm dynamically adjusts the number of motion models being used, striving to select the simplest configuration that still accurately fits the event data. The λ_M parameter acts as a regularization factor, penalizing the use of too many motion models, which encourages the segmentation to converge on the correct number of IMOs by avoiding overfitting with unnecessary labels.

4.1.5 Optimization

The overall algorithm iteratively alternates between two steps until it converges to a solution that minimizes the energy function. The two key steps in this iterative process are:

1. **Updating Event-Cluster Assignments (L):** In this step, the motion model parameters (M) are held constant. The goal is to find the optimal assignment of events to the various motion clusters that minimize the overall energy. The optimization is efficiently carried out using the α-expansion based graph-cut method, which is well-suited for handling energy minimization problems involving the Potts model and label costs. The algorithm iterates through different labels (α), and in each iteration, it employs a minimum $s-t$ cut to partition the events into two groups: one associated with the current labelling and the other with the α label. The α-expansion move is then executed based on the $s-t$ cut that leads to the lowest energy configuration. This process is repeated until convergence, resulting in the optimal labelling configuration for the given set of motion models, [48].
2. **Updating Motion Models (M):** In this step, the event-cluster assignments (L) are kept fixed, and the focus shifts to refining the parameters of the motion models for each cluster. Since the labels are fixed, the motion models can be optimized independently for each cluster. The original motion compensation scheme is utilized here, where the objective is to identify the motion model parameters that maximize the contrast or sharpness of the IWE for the events belonging to each cluster. The optimization is performed separately for each motion model, leading to refined motion parameters that better capture the motion of the events assigned to each cluster, [48].

The final output is a segmentation of the events into distinct motion clusters and a set of refined motion models that accurately represent the motion of the objects or the background in the scene.

4.1.6 Initialization

The optimization procedure requires an initial set of motion models. The initialization step provides this initial set of motion models, which serves as the starting point for this iterative optimization process. The process involves the following steps:

1. **Subdivision:** The space-time volume encompassing the events is systematically divided into multiple sub-volumes. The number of sub-volumes increases with the level of subdivision, enabling the algorithm to capture motion at various scales, from large objects to smaller ones, [48].
2. **Motion compensation:** The motion compensation scheme is applied to each of these sub-volumes. This involves warping the events within each sub-volume according to a motion model and aggregating them into an Image of Warped Events (IWE). The sharpness or contrast of the IWE serves as an indicator of how well the events align with the assumed motion model, [48].
3. **Model pool creation:** For each sub-volume, the motion model that yields the sharpest IWE is selected as a potential candidate representing the motion within that region. The collection of these candidate models from all the sub-volumes constitutes the initial model pool, [48].

4.2 Neural Network Approach for Depth and Motion Estimation

This section explores a motion segmentation approach utilizing neural networks. A diverse array of architectures is employed, including adaptations of established models [25, 47] and a novel design, to comprehensively evaluate neural networks' efficacy within this domain.

4.2.1 Problem Formulation

This method aims to estimate per-pixel motion and segment objects from monocular images by leveraging depth and ego-motion information. The approach utilizes three neural network architectures to predict depth and pose, which are then combined with the camera's ego-motion to produce an ego-motion flow. This flow is used to segment objects in the scene and estimate their per-pixel motion, providing a comprehensive understanding of the dynamic interactions within the environment.

4.2.2 Data Acquisition & Processing

The input data consists of grayscale images generated from event streams captured by an event-based camera. Each image represents the accumulated events within a specific time slice, with pixel intensity indicating the number of events at that location. Alongside these images, corresponding depth maps provide ground truth depth information for each pixel in the scene. This combination of intensity and depth data serves as the input for subsequent neural network processing.

4.2.3 Architectures & Implementation

The input image, reference images and the respective ground truth image are collectively provided to the networks in the form of 2D maps, as shown in fig 4.3. Three distinct neural network architectures are employed, each performing depth and pose estimation. These networks are not employed sequentially but rather evaluated independently to assess their performance:

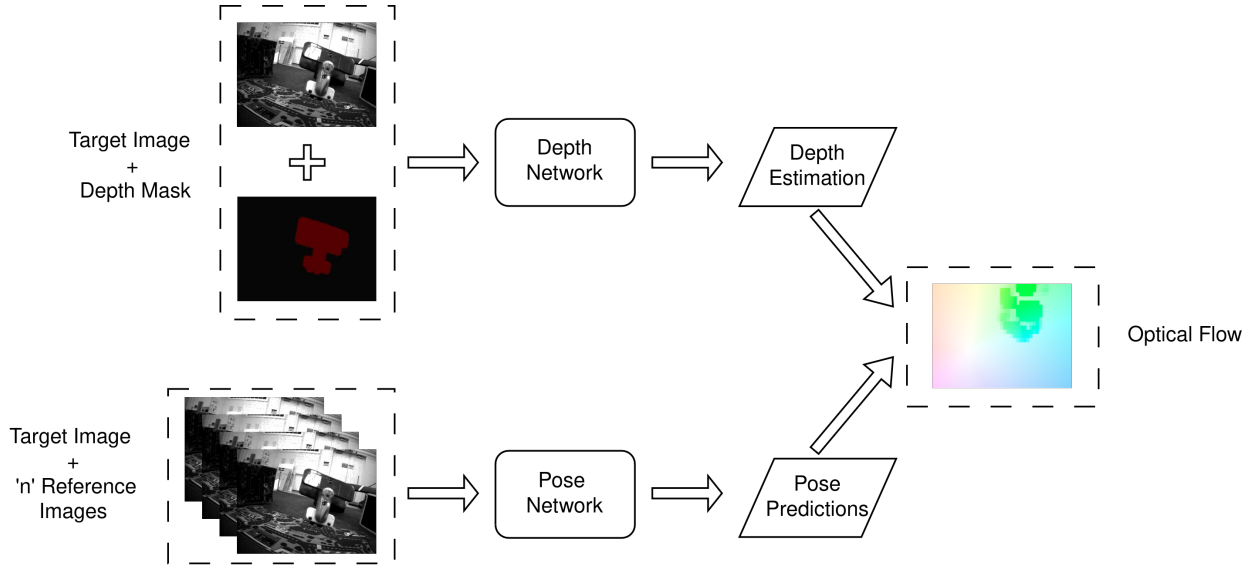


Figure 4.3: A depth network is trained with supervision to predict scene depth. Meanwhile, a pose network processes sequential event slices to create a pixel-wise pose mixture model, yielding both pose mixtures and their associated probabilities. By combining the outputs from these two networks, optical flow is generated [25].

4.2.3.1 Evenly Cascaded Network (ECN)

The Evenly Cascaded Network (ECN), as introduced in [47], follows an encoder-decoder structure for depth and pose estimation. The network architecture is built around cascaded convolutional blocks that incorporate residual learning, ensuring that both high-level semantic information and fine-grained details are retained throughout the network, as illustrated in Fig 4.4.

Key Features:

- Cascade Mechanism: The core of the ECN is its cascading structure, which blends input feature maps with processed output across layers. This mechanism ensures spatial consistency, crucial for tasks like depth and pose estimation.
- Residual Learning: Residual connections are employed within the cascade mechanism, continuously refining predictions for both depth and pose.
- Convolutional Blocks: ECN leverages both single and double convolutional blocks, with ReLU activations, feature normalization, and pooling mechanisms to adjust feature map resolutions.
- Normalization: The network is adaptable to various batch sizes and computational constraints by supporting multiple normalization types (supporting BatchNorm, InstanceNorm, GroupNorm, FeatureDecorr).
- Initialization and Pooling: Xavier initialization stabilizes training, and adaptive average pooling adjusts feature map resolutions to maintain consistency between different layers.

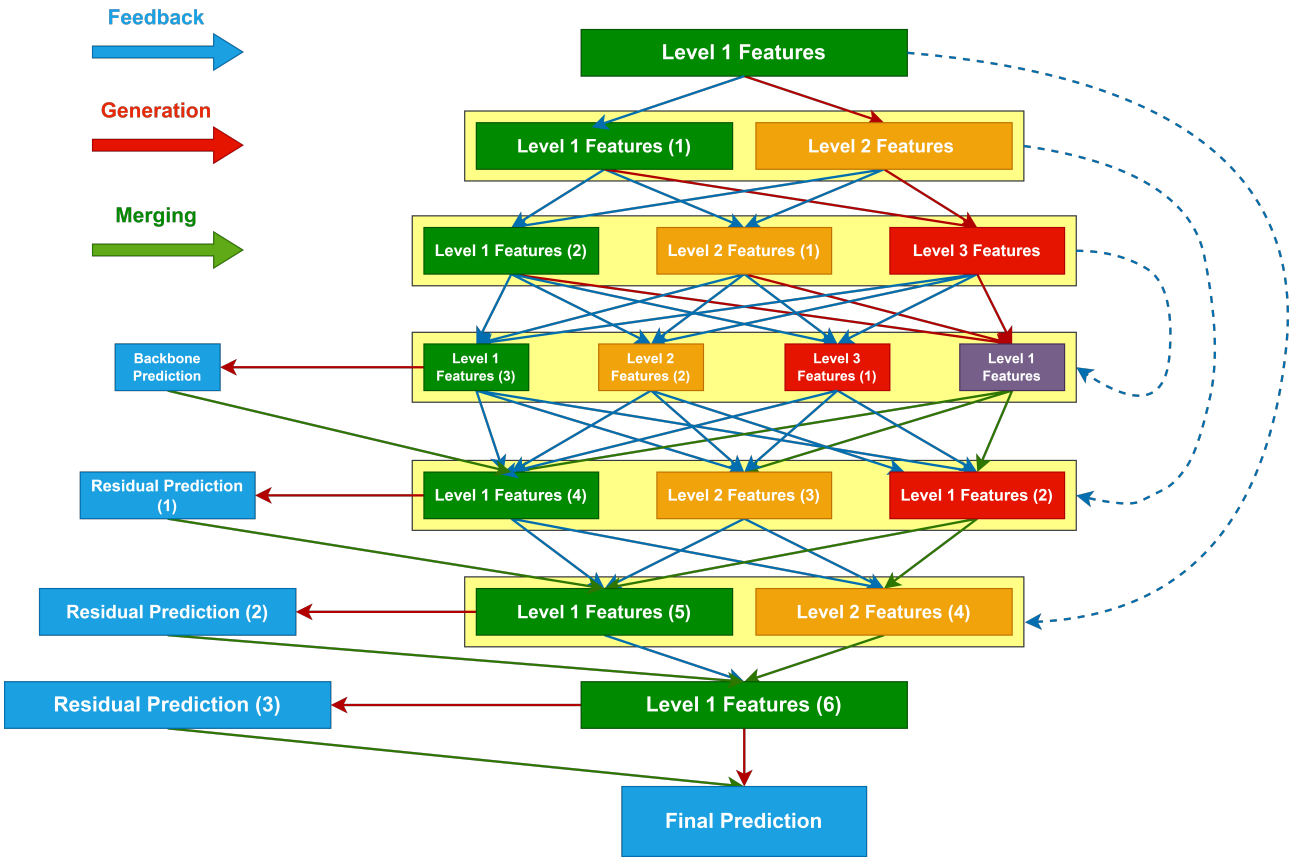


Figure 4.4: **Evenly Cascaded Network**: This diagram illustrates the working principle of the cascading network. The network employs a multi-level feature extraction and refinement process. Features are extracted at different levels of abstraction, and feedback signals are incorporated to improve feature quality. The final prediction is generated through a series of refinement stages, combining features from multiple levels. [47].

The architecture [47], starts by accepting a stacked input of target and reference images. These are processed through initial convolutional layers that reduce spatial dimensions while expanding the feature space. The encoded features are then passed through several $CascadeLayer$ modules, which scale down feature maps using predefined factors (e.g., 0.5) while aggregating multi-level information.

Each CascadeLayer consists of a double convolution block where each convolution is followed by batch normalization and $ReLU$ activation. The cascade mechanism combines the output of each convolution block with the input feature map, handling mismatched feature dimensions through concatenation or element-wise addition. Pooling mechanisms like adaptive average pooling ensures that feature maps maintain consistent resolutions across layers.

In the decoding phase, feature maps are passed through $InvertedCascadeLayer$ modules. These layers mirror the encoding process but recover spatial resolution progressively. Feature maps from the encoding phase are combined with outputs of previous decoding layers, ensuring that high-level features are retained while reconstructing finer details.

The final stage of the ECN generates either depth or pose predictions. A sequence of disparity maps is produced in the depth network, each refined using residual learning to achieve more accurate predictions. In the pose network, the output is a 6-DoF pose prediction for each reference image, derived from the final feature maps.

4.2.3.2 Encoder-Decoder Network (EDN)

The Encoder-Decoder Network (EDN) as developed in [25], structured as a convolutional neural network and inspired by the U-Net architecture, employs the ECN as its backbone to effectively estimate depth, pixel-wise motion segmentation, and 6-DoF ego-motion from monocular event data. This integrated learning pipeline extends its capabilities to encompass per-segment linear velocity estimation.

Key Features:

- Encoder-Decoder Structure: The model employs a classic encoder-decoder architecture, where the encoder progressively downsamples the input to extract high-level features, and the decoder symmetrically upsamples to generate the final predictions.
- Convolutional Backbone: The encoder utilizes a series of convolutional layers with increasing channel depths, capturing hierarchical features from the input data.
- Depth Prediction: Disparity maps are predicted using a dedicated layer at different stages in the decoder. Each disparity map is progressively refined using the up-sampled previous disparity predictions. These maps are later converted to provide the depth information.
- Pose Prediction: To predict the 6-DoF camera pose for each reference image, a final convolutional layer outputs a 6-dimensional vector. This vector includes both rotation and translation components, providing a detailed representation of the camera's movement relative to the reference frames. These pose estimates are essential for understanding the camera's orientation and position within a 3D space.
- Optional Decoder Outputs: The decoder can be configured to produce multiple outputs, including:
  1. Explanation Masks: Masks indicating regions corresponding to different motion explanations (e.g., background vs. moving objects).
  2. Pixel-wise Poses: Detailed pose information at the pixel level.
  3. Disparity: Disparity maps, which can be converted to depth information.
- Skip Connections: Similar to U-Net, skip connections bridge the encoder and decoder at corresponding levels, facilitating the flow of fine-grained spatial information to the decoder.
- Transposed Convolutions: The decoder employs transposed convolutions for upsampling, enabling the recovery of spatial resolution.
- Sigmoid and Scaling: Sigmoid activation is used in conjunction with scaling factors (alpha and beta) to constrain the output range of disparity predictions.
- Weight Initialization: Xavier initialization is applied to convolutional and transposed convolutional layers to promote stable training.

The learning pipeline is divided into two networks, each running in parallel:

1. **DispNetS:**

   The following Fig 4.5 illustrates the network's design. The encoder (down-sampling path) in the network consists of seven layers of convolution, with each layer progressively increasing the number of feature channels. This architecture compresses the input image into a compact feature representation. The convolution operations, performed by a down-sampling convolution function, use a stride of 2 to reduce the spatial resolution by half at each layer, followed by ReLU activations to introduce non-linearity. The kernel sizes start at 7x7 in the first layer and decrease to 3x3 in the subsequent layers. As the image is down-sampled, the feature channels expand from 32 to 512, allowing the network to capture both local and global image details.

   Once the bottleneck is reached in the encoder, the network begins to upscale the feature maps back to the original resolution using transposed convolutions, carried out by an up convolution function. During this process, the higher-level features are combined with the corresponding lower-level features from the encoder through skip connections. This helps in preserving finer spatial details that might have been lost during down-sampling, making the up-sampling process more effective in recovering the original image resolution.

At various stages in the decoder, disparity maps are predicted, which are then used to compute the depth of the scene. Although disparity predictions are made at multiple scales, only the final up-sampled output is utilized. A dedicated function is responsible for predicting these disparity maps, applying a single convolution followed by a sigmoid activation, which constrains the predicted values between 0 and 1. These disparity values are then scaled using the alpha and beta parameters to generate realistic depth values.

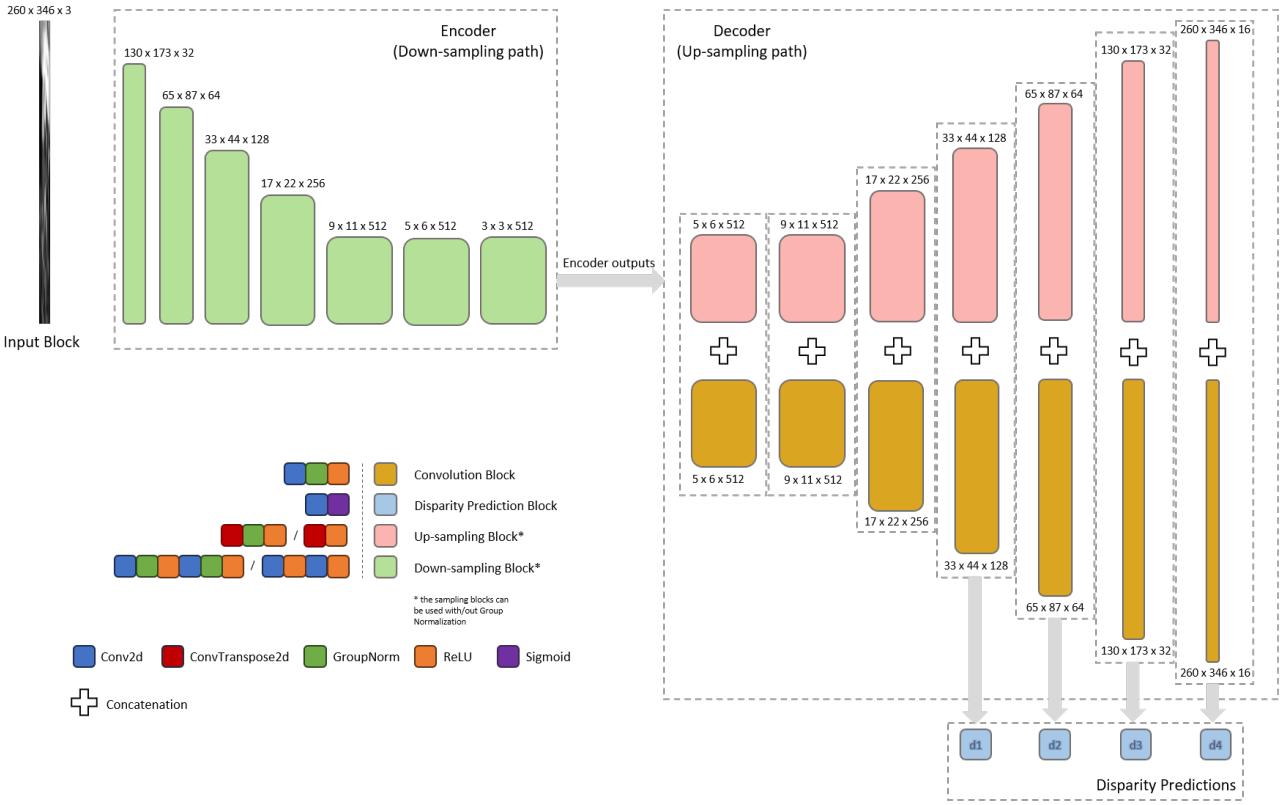


Figure 4.5: **DispNetS Architecture**: The network's encoder down-samples the input through seven convolutional layers, increasing feature channels. The decoder up-samples using transposed convolutions with skip connections. Final disparity maps are used for depth estimation [25]

In the forward pass, the input image moves through seven convolutional layers, progressively down-sampling and expanding the feature maps. After reaching the bottleneck, the feature maps are up-sampled using transposed convolutions. Throughout the up-sampling process, skip connections are used to concatenate feature maps from the corresponding encoder layers to preserve spatial details. The final output is a predicted disparity map at multiple resolutions, with the up-sampled disparity matching the original input resolution.

2. **PoseExpNet:**

Similar to the DispNetS, the PoseExpNet (as shown in Fig 4.6) encoder uses several convolutional layers to compress the input into a feature representation. However, instead of a single input image, PoseExpNet processes a stack of images, which typically includes a target image and multiple reference images (e.g., consecutive frames from a video). The feature channels progressively increase from 16 to 256 as the input moves through deeper layers, capturing relevant motion information needed for pose prediction. It provides a global pose estimate, as well as localized pose estimates at different resolutions.

At the bottleneck layer, PoseExpNet employs a 1x1 convolutional layer to predict the 6-DoF camera pose for each reference image. This pose consists of both rotation and translation vectors, providing a comprehensive representation of the camera's movement relative to the reference frames. These pose estimates are key for understanding the camera's orientation and position in a 3D space.

In addition to predicting the camera pose, PoseExpNet also generates an explainability mask. This mask highlights areas in the image where pose estimation may be unreliable, which often occurs due to dynamic objects or occlusion. The explainability mask is predicted at multiple resolutions and refined

through up-sampling layers. This mask is useful for identifying regions where pose predictions should be treated with caution.

PoseExpNet can also predict disparity maps at various scales, though this feature is generally redundant when used alongside DispNetS. Disparity prediction in PoseExpNet is optional and often not required when both networks are combined, as DispNetS is specifically tailored for that task.

During the forward pass, PoseExpNet processes the stacked input images (target and reference images) through six convolutional layers, compressing the spatial information. At the bottleneck, the network predicts the relative camera motion in the form of a pose. Additionally, an explainability mask may be predicted, indicating regions where pose estimates may be inaccurate. The network uses transposed convolutional layers for up-sampling, and finally, it returns the predicted pose, explainability mask, and optional disparity maps.

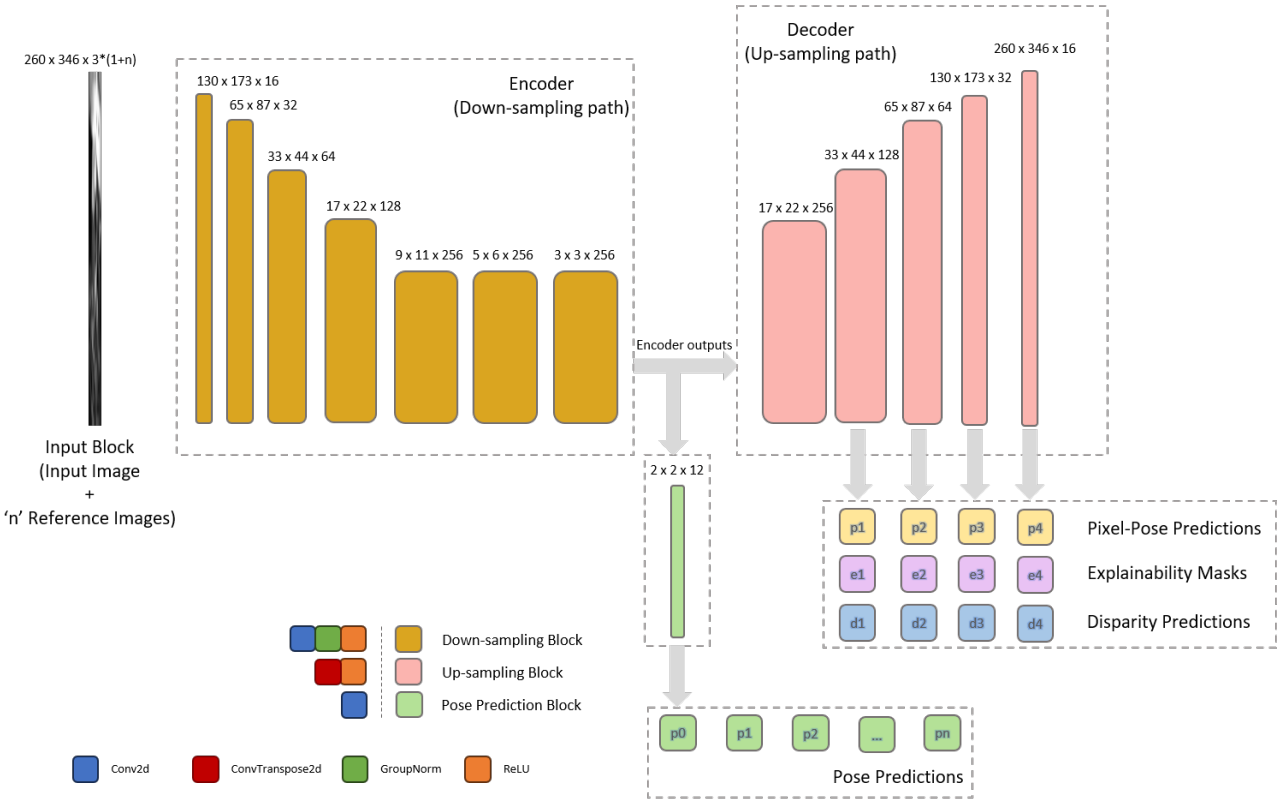


Figure 4.6: **PoseExpNet Architecture**: The network processes stacked images through convolutional layers to predict 6-DoF camera pose and generates an explainability mask to highlight unreliable regions. Optional disparity maps can also be predicted [25]

4.2.3.3 ResNet18 Encoder-Decoder Network (REDN)

This approach integrates ResNet as its backbone network to leverage its advanced feature extraction capabilities. In this approach, ResNet18 is employed as a feature extractor due to its robust performance in capturing intricate details from the input images.

Key Features:

- Feature Extraction: The ResNet18 is employed as the encoder to extract hierarchical features from the input images. This deep network is particularly adept at retaining detailed spatial information while reducing computational overhead by freezing its parameters and focusing on the prediction layers for further learning.
- Depth, Pose, and Explainability Estimation: Both models together predict disparity maps, pose estimation (6-DoF camera motion) and optional explainability masks for identifying unreliable motion or depth

predictions.

- Multiscale Prediction: Predictions for depth, pose, and explainability are made at multiple scales to capture both global and local details.
- Skip Connections: Feature maps from the encoder are combined with up-sampled decoder maps to retain spatial details and improve accuracy.

Again, we divide the ResNet18 Encoder-Decoder Network (REDN) network into two parts, one for depth estimation and the other for pose estimation:

1. **ResDepthNet:**

   The network uses a pre-trained ResNet18 model as the encoder. The encoder is initialized with pre-trained weights and uses the initial layers of ResNet18, including the first convolutional layer, batch normalization, ReLU, and max-pooling, followed by the first two residual blocks (ResNet18 layers 1 and 2). This allows the network to learn rich feature representations from the input image.

   The decoder is responsible for up-sampling the compressed feature maps back to the original input resolution. This is done using transposed convolutional layers, which increase the spatial resolution by a factor of two at each layer. Each up-sampling step is followed by a convolution layer that refines the feature maps.

   The network predicts disparity maps at four different scales. The disparity maps are generated, which apply a convolution followed by a sigmoid activation to predict disparity values between 0 and 1. These predictions are then scaled by the parameters alpha and beta to yield meaningful depth values.

   The input image passes through the encoder, which extracts feature maps. These feature maps are then progressively up-sampled through the decoder. Skip connections ensure that spatial details from the encoder are retained. Disparity maps are predicted at multiple resolutions, with the final one up-sampled to match the input image size. The network is visualised in Fig 4.7.

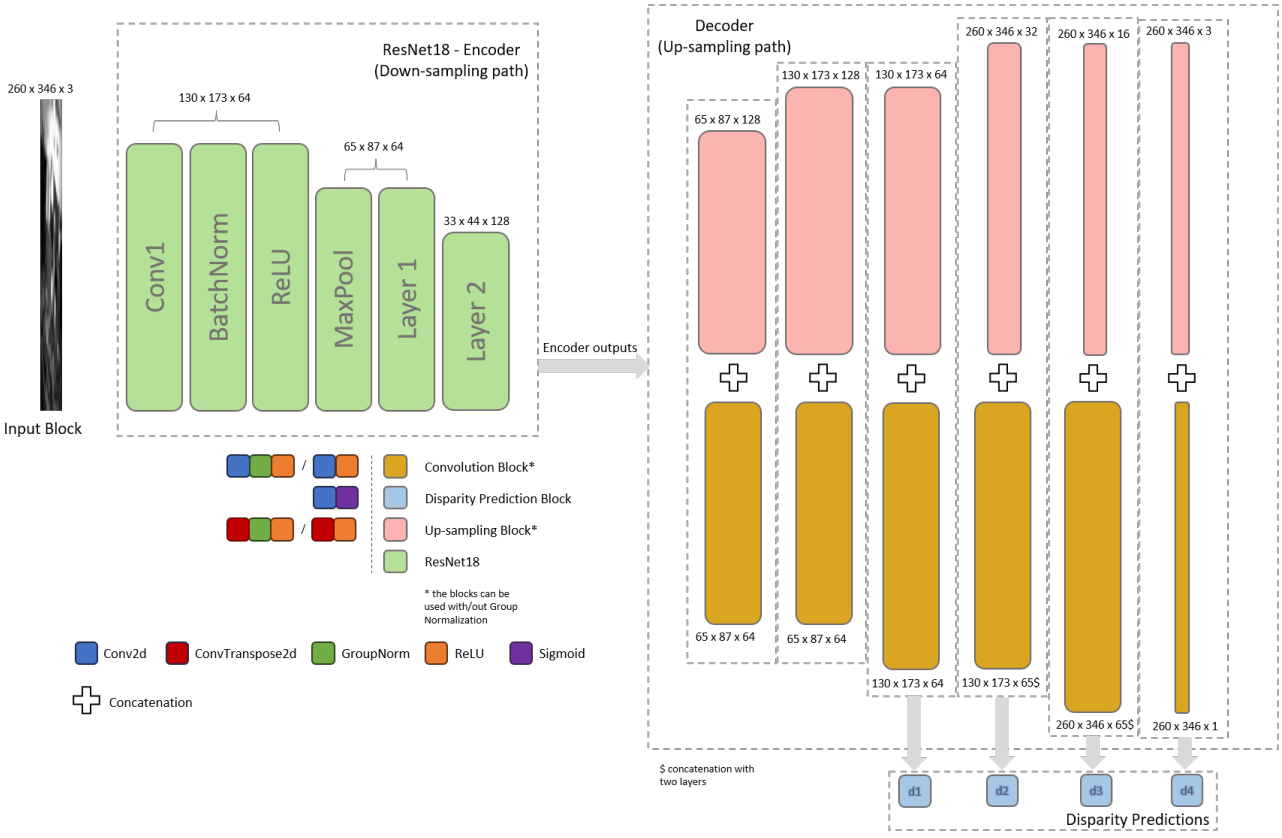


Figure 4.7: **ResDepthNet Architecture**: The network uses a pre-trained ResNet18 as the encoder, followed by a decoder that up-samples feature maps using transposed convolutions. Disparity maps are predicted at multiple scales and refined with skip connections for depth estimation

2. **ResPoseNet:** The ResPoseNet also uses ResNet18 as the encoder. It takes a stack of images (target image + reference frames) as input. The ResNet18 layers extract feature representations from these images.

   The network predicts the relative camera pose for each reference image. This is achieved by applying a 1x1 convolution layer to the upsampled feature maps. The network outputs a 6-DoF pose. Simultaneously, the explainability mask is predicted at different scales using convolutional layers and upsampling operations.

   ResPoseNet can optionally predict disparity maps at different resolutions, providing depth information, similar to its counterpart ResDepthNet. The predicted disparity maps are scaled by the parameters alpha and beta to generate realistic depth values.

   The network (as shown in Fig 4.8) processes the input image stack (target image + reference frames) through the encoder. The extracted features are then passed through a series of upsampling layers (upconv) to restore spatial resolution. Depending on the configuration, the network outputs camera poses, explainability masks, and disparity maps.

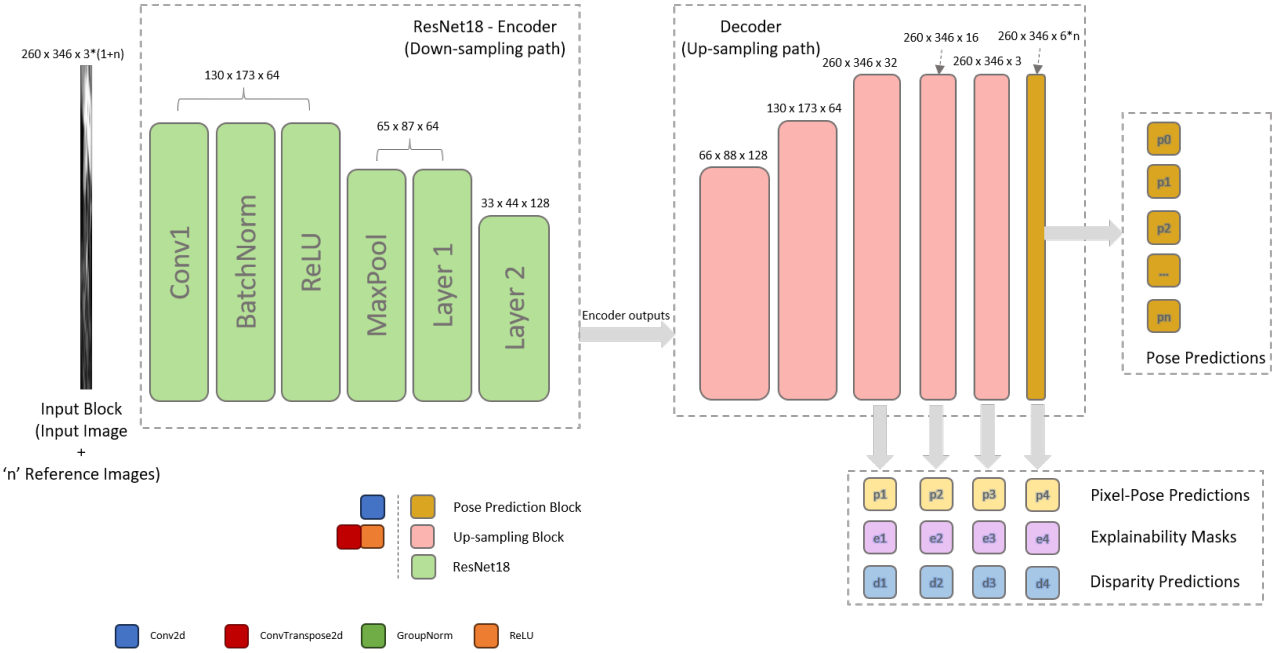


Figure 4.8: **ResPoseNet Architecture**: Using ResNet18 as the encoder, ResPoseNet processes stacked images to predict 6-DoF camera pose and generate explainability masks. Optionally, disparity maps are predicted at multiple resolutions for depth estimation

4.2.3.4 Color Scheme for Depth and Motion Outputs

As the employed models predict the depth and pose estimations in the form of 2D images depicting the depth, direction and magnitude of motion, a consistent colour scheme is systematically employed across all models to visualize the outputs. This standardized colour coding enhances the clarity of the results and facilitates a more comprehensive understanding of the models' performance. The subsequent sections outline the specific details of the colour schemes utilized:

1. **Depth Estimation Outputs:** The outputs of depth estimation, typically utilize a gray scale colour map, where brightness levels correspond to object proximity relative to the camera. Brighter areas in the visualization represent closer objects, while darker shades indicate greater distances from the camera. Consequently, regions rendered in white or light gray signify proximity, whereas black or dark gray areas denote objects that are farther away.
2. **Optical Flow Outputs:** For the optical flow outputs, it employs a colour wheel to visually represent the direction and magnitude of motion. Each colour on the wheel corresponds to a specific direction of movement, while the intensity or brightness of the colour reflects the magnitude of that motion [11].

Tables 4.1 and 4.2 provide an overview of how several common colors represent the direction of motion for various objects, while the brightness levels convey the velocity of that motion.

Color	Direction
Red	Right
Green	Upward
Blue	Left
Cyan	Downward
Magenta	Upper Right
Purple	Upper Left
Orange	Lower Right
Yellow	Lower Left

Table 4.1: Direction Representation in Optical Flow

Brightness Level	Magnitude
Bright	High
Medium	Medium
Dark	Low

Table 4.2: Magnitude Representation in Optical Flow

4.2.4 Loss Functions

During each forward pass of the training process, multiple loss components are calculated to evaluate the performance of the depth and pose networks. These losses are combined to form the total loss, which is then used to update the model parameters [25].

4.2.4.1 Photometric Loss

The photometric loss is calculated by comparing the target image with the warped reference images. The model predicts depth from the target image and the relative pose between the target and reference images. The reference images are warped using the predicted depth and pose, and the photometric loss is calculated as the difference between the warped reference images and the target image. The goal is to minimize this difference, ensuring that the predicted depth and pose accurately align the reference images to the target image. The photometric loss $L_{\text{photometric}}$ [25], is defined as:

$$L_{\text{photometric}} = \frac{1}{N}\sum_{i=1}^{N}(\|I_i - I_i^{'}\|_1 + \lambda_{\text{ssim}} \cdot \text{SSIM}(I_i, I_i^{'})) \tag{4.1}$$

where I_i is the target image, $I_i^{'}$ is the warped reference image, and λ_{ssim} is the weight for the SSIM loss term.

Structural Similarity Index (SSIM) [30], is a metric used to measure the similarity between two images. It considers changes in structural information, luminance, and contrast:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.2}$$

where:

- μ_x and μ_y are the mean values of x and y, respectively.
- σ_x^2 and σ_y^2 are the variances of x and y, respectively.
- σ_{xy} is the covariance of x and y.
- C_1 and C_2 are small constants to avoid division by zero.

4.2.4.2 Explainability Loss

The explainability loss is introduced to handle areas where photometric reconstruction may not be reliable, such as occlusions or moving objects. The model predicts an explainability mask, which identifies regions of the image where reconstruction is less reliable. The explainability loss penalizes large or incorrect masks, encouraging the model to use the mask only where necessary. This ensures that the model does not over-rely on masking and can still learn meaningful depth and pose information in most image regions [25]. It is described as:

$$L_{\text{explainability}} = \frac{1}{N} \sum_{i=1}^{N} \|M - M_{\text{target}}\|_1 \tag{4.3}$$

where M is the predicted explainability mask and M_{target} is the target explainability mask.

4.2.4.3 Smoothness Loss

The smoothness loss enforces smooth transitions in the predicted depth or optical flow across neighbouring pixels. This helps the model avoid sharp discontinuities in the depth map, especially in regions where depth information is sparse. The smoothness loss encourages the depth map to have a piecewise smooth structure, with large variations in depth only allowed at object boundaries [25]. It is given as:

$$L_{\text{smoothness}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\|\nabla_x D_i\|_1}{\|\nabla_x D + \epsilon\|_p} + \frac{\|\nabla_y D_i\|_1}{\|\nabla_y D + \epsilon\|_p} \right) \tag{4.4}$$

where $\nabla_x D$ and $\nabla_y D$ are the gradients of the depth map D in the x and y directions, respectively, and ϵ is a small constant to avoid division by zero.

4.2.4.4 Pose Loss

The pose loss penalizes incorrect or extreme camera motion predictions. As the pose network estimates the relative pose between the target and reference images, large or implausible camera motions can negatively affect the model's ability to reconstruct the scene. The pose loss ensures that the predicted camera motion stays within reasonable bounds [25], and is depicted as:

$$L_{\text{pose}} = \frac{1}{M} \sum_{j=1}^{M} \|\text{Pose}_{\text{pred},j} - \text{Pose}_{\text{target},j}\|_2 \tag{4.5}$$

where $Pose_{pred,j}$ is the predicted pose and $Pose_{target,j}$ is the target pose.

4.2.4.5 Combined Loss

Finally, the total loss is computed as a weighted sum of the individual loss components. Each component (photometric, explainability, smoothness, and pose losses) is multiplied by its corresponding weight.

$$L_{\text{total}} = \lambda_{\text{photometric}} L_{\text{photometric}} + \lambda_{\text{explainability}} L_{\text{explainability}} + \lambda_{\text{smoothness}} L_{\text{smoothness}} + \lambda_{\text{pose}} L_{\text{pose}} \tag{4.6}$$

where $\lambda_{photometric}$, $\lambda_{explainability}$, $\lambda_{smoothness}$, and λ_{pose} are weights for each loss component.

4.2.5 Best Checkpoint

During the training phase, both the training and validation losses are monitored. The model architecture is specifically designed to minimize total loss, thereby ensuring optimal performance. The best-performing

model is identified by saving the checkpoint corresponding to the lowest recorded total loss. This total loss is calculated as the sum of the losses from the two sub-networks responsible for the depth and pose estimation.

4.2.6 Backward Pass

After the forward pass, where the network computes predictions and the corresponding losses are calculated, the training process transitions into the backward pass. In this phase, gradients are computed and used to update the network's parameters, ensuring that the model learns from its errors and improves over time.

4.2.7 Validation

In the validation phase, predictions are generated on the validation set. Key metrics such as IoU, Dice, and Pixel Accuracy are computed, and validation losses are calculated. These metrics and losses are logged to track the model's progress, while visualizations of the validation results provide further insights into the model's performance.

4.2.8 Evaluation

The model's performance is evaluated using a separate test set that is independent of the training and validation data. This process provides an objective assessment of how well the model generalizes to unseen data.

5 Datasets

This chapter presents a concise overview of the published datasets referenced in [48] and a custom dataset specifically developed for this study. These datasets are specifically designed to cater for the needs of event camera-based research.

5.1 DVSMOTION20 Dataset

The DVSMOTION20 dataset [5], consists of four real indoor sequences: checkerboard, classroom, conference room, and conference room translation. The dataset was collected using an IniVation DAViS346 camera. This camera boasts a spatial resolution of 346×260 pixels, outputs frames at up to 60 frames per second, and records events with microsecond precision, along with 6-axis IMU data at approximately 1 kHz. Each scene is captured for approximately 13 to 16 seconds, with the initial 3 seconds dedicated to the IMU calibration. The following 7 to 8 seconds of DVS data are used for performance evaluation. Each recorded data file is approximately 500 MB in size.

The sequences are primarily focused on camera motion within stationary scenes, all but the conference room translation sequence involve rapid movements with complex, random camera motions, presenting significant challenges for optical flow methods, which often yield normal flow results. In the conference room translation sequence, the motion is restricted to horizontal (yaw rotation), enabling us to assess the hypothesis that optical flow methods generally perform better with simpler pixel motion.

In addition to the primary sequences, DVSMOTION20 features supplementary sequences titled "hands" and "cars", which capture multiple object motions in addition to camera movement. These object motions are spatially local and are defined by strong motion boundaries, which result in scenarios that involve motion occlusion. Visual inspection of the estimated motion vectors is effective for identifying optical flow failures, particularly in instances of occlusion or substantial object movements, even when ground truth motion vectors cannot be inferred from the IMU.

5.2 Extreme Event Dataset (EED)

The Extreme Event Dataset (EED) [27], used in this study was collected with the DAVIS sensor under two scenarios: mounted on a quadrotor and in a hand-held setup to capture a range of non-rigid camera motions. The recordings feature objects of various sizes moving at different speeds under diverse lighting conditions, which showcase the pipeline's ability to detect at high rates, including sequences where tracked objects change speed abruptly. The dataset contains over 30 recordings, with the "Strobe Light" sequence as a highlight, where a flashing light in a dark room creates significant noise while another quadrotor moves within the space.

The dataset was acquired using the DAVIS240B bio-inspired sensor with a 3.3 mm lens, offering a horizontal and vertical field of view of 80°. Most sequences were captured in a hand-held setting, while the quadrotor sequences were collected using a modified Qualcomm Flight platform to connect the sensor to an onboard

computer. The fully loaded quadrotor weighs about 500 grams and features a Snapdragon ARM CPU with four cores running at up to 2.3 GHz.

The recordings are organized into several sequences based on scenario types, featuring a variety of camera motions like "Fast Moving Drone" which has a sequence of a small quadrotor moving across various backgrounds in daylight, "What is a Background?" that demonstrates the ability to track an object placed behind a net, with motion visible only through the net, and many such sequences.

5.3 EV-IMO Dataset

The EV-IMO Dataset [25], is an event camera dataset, which is designed to capture scenarios involving multiple independently moving objects and high-speed camera motion. It offers accurate depth maps, object masks, and object trajectories, recorded at over 200 frames per second. The dataset includes over 30 sequences, amounting to about 30 minutes of data.

The dataset uses DAVIS event cameras to generate accurate event data. A 3D scanning process is also applied for the objects and room, combined with the VICON motion capture to track both objects and camera movement during recording. This system allows the dataset to generate accurate depth and object masks by projecting 3D point clouds onto the camera plane. The use of high-speed motion tracking provides ground truth data that is rather difficult to capture with traditional depth sensors.

Each object is equipped with reflective markers for tracking and scanned using a high-resolution 3D scanner. This allows for precise depth mapping and trajectory tracking of both the camera and objects. Additionally, the dataset computes camera and object velocities, making it valuable for evaluating motion detection and object tracking algorithms.

The dataset consists of sequences categorized by background types (e.g., table, plain wall, floor) and includes cluttered scenes with independently moving objects. This diversity in background and motion is crucial for event-based cameras, which capture only edge information.

5.4 Custom Dataset

This dataset was collected using the DVXplorer event camera in a hall under ambient lighting, prepared to capture several motion scenarios involving both human subjects and objects. The camera has a resolution of 640×480 pixels (VGA), enabling detailed event capture. With a latency of less than 5 microseconds, it is highly responsive to rapid movements. The temporal resolution reaches 1 microsecond, allowing precise tracking of subtle motion changes. Additionally, the camera boasts a dynamic range of over 120 dB, making it robust in challenging lighting conditions.

The recordings include sequences of a person moving with and without an object, capturing diverse motion patterns. The dataset encompasses various situations such as stationary cameras with object movement, slow camera motion alongside object movement, and fast camera movement with simultaneous object motion. These variations provide a comprehensive dataset for studying motion detection, object tracking, and event-based scene analysis under real-world conditions.

6 Experiments and Results

This chapter covers the experiments conducted using both the space-time graph-based method and the neural network-based approach. We'll compare the performance of these models and analyze the results. First, we'll review the various experimental setups and parameter tuning, followed by a summary of the outcomes.

6.1 Motion Segmentation using Energy Minimization

We began by evaluating the energy minimization approach to assess its suitability for our study. Initially, we replicated the implementation on the published datasets, as described in Chapter 5. After reproducing the results, we tailored the algorithm to our custom dataset for a more in-depth analysis. This allowed us to explore further the effectiveness of the method proposed in the study [48] and to understand its potential in our specific application. For this experiment, the input data provided to the model consisted of tuples in the form of (timestamp, x, y, polarity), capturing the event-based information of each scene. Each scene spanned 1-2 seconds, and the corresponding event data was fed into the model. The input data structure remained consistent throughout, with each tuple containing essential spatial and temporal information. Initially, key parameters influencing the model's performance were fine-tuned according to the spatial resolution and dynamics of each scene, after testing on the ideal values provided in [48]. The output was generated in the form of accurately segmented 2D images, as presented in Table 6.1.

6.1.1 DVSMOTION20 Dataset

The first experiment was conducted using the DVSMOTION20 dataset, focusing specifically on the "hands" sequence. In this scene, since there was no camera motion and only hand movements were present, the parameter $num\_events\_per\_image$, which defines the number of events processed in the space-time volume was set to 15,000 to capture the relevant motion. This value was chosen based on the optimal range of 15,000 to 30,000, as suggested in the study [48]. For the model's regularization terms as described in the subsection 4.1.4, the λ_D parameter, responsible for data fidelity, was initialized at 1. To control the smoothness in segmentation, the λ_P parameter was initially set to 20 and progressively increased to 30 to balance over-segmentation and accuracy, while the λ_M parameter associated with the MDL term was initially set to 0. It was gradually adjusted starting from 1,000, increasing by increments of 1,000, and eventually stabilized at 8,000 as the λ_P value became more refined.

The parameters related to the MRF model were configured as follows:
$NUM\_ITER\_SEGMENTATION$, which determines the number of iterations for continuous updates, was set to 5. This ensures that the model iteratively refines the segmentation with each update. Additionally, $NUM\_ITER\_LABELING$, which handles discrete updates using alpha-expansion graph cuts, was set to 2, following the guidelines provided in [48]. These values remained constant throughout the experiments. For initialization, the $HybridModelCombination$ parameter was set to use the 4D model exclusively, as it offered the most comprehensive representation of both spatial and temporal dynamics required for accurate motion segmentation in this context. The spatial division parameters, $BracketRow$ and $BracketCol$, were both set to 8, ensuring efficient partitioning of the space-time volume. This value was determined based on the number of objects present in the scene and their relative distance from the camera. Using these settings resulted in

successful segmentation of the moving objects, as shown in the top row of Table 6.1. Multiple motion regions were detected on the same object, likely due to the model's tendency to over-segment the object. This led to an unnecessary division of what should be a single cohesive motion into smaller, separate segments.

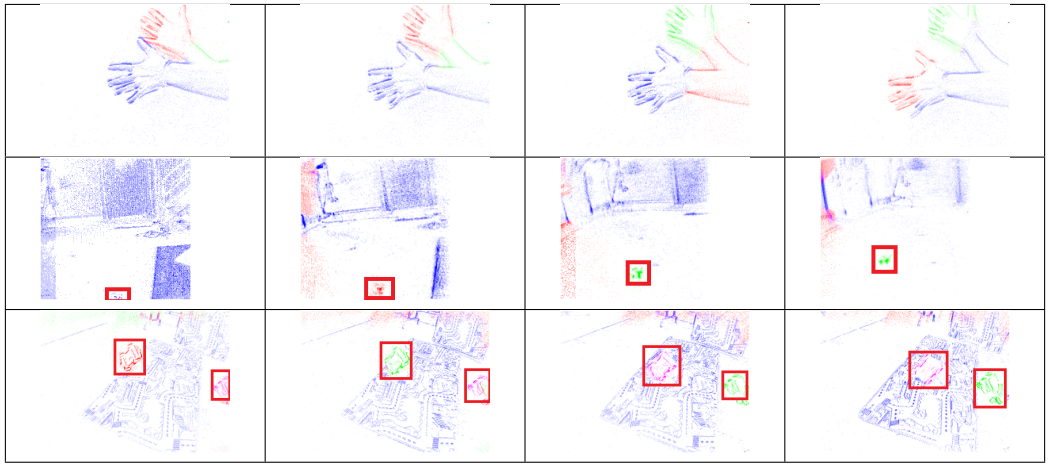


Table 6.1: Segmentation results on the DVSMOTION20, EED, and EV-IMO datasets. Time runs from left to right. The ground truth bounding box (in red) indicates the manually annotated 2D location of the IMOs.

6.1.2 Extreme Event Dataset (EED)

The model was further evaluated on the EED dataset, specifically focusing on the "fast_drone" scenario. This scenario involved a drone rapidly navigating through a room while the camera tracked its movement. Given the higher number of events in this instance, the following parameters were adjusted: $num\_events\_per\_image$ was set to 20,000, λ_D to 1, and λ_P to 8,000, while other parameters remained unchanged from the previous tests. With these adjustments, the segmentation results significantly improved. The drone, characterized by its velocity, was distinctly grouped into a separate cluster from the background, influenced by the camera's movement. The segmentation, visualized in Table 6.1 (middle row), uses colour coding to illustrate this distinction: the drone is highlighted in green, while the background's motion, caused by the camera, is represented in red and blue. This suggests that the model effectively differentiated and clustered various IMOs according to their respective velocities.

6.1.3 EV-IMO Dataset

The third set of experiments was conducted using the EV-IMO dataset, which features two objects in a collision and the camera's ego motion. The parameters were set as follows: $num_events_per_image$ = 30,000, λ_P = 25, and λ_M = 8,000 adjusted according to the scene dynamics, while all other parameters were kept constant. In this sequence, three distinct motions are observed: two IMOs (independent moving objects) and the camera's movement, as illustrated in Table 6.1 (bottom row). The two objects were differentiated by their respective velocities, while the background table exhibited varying velocities due to the camera's motion.

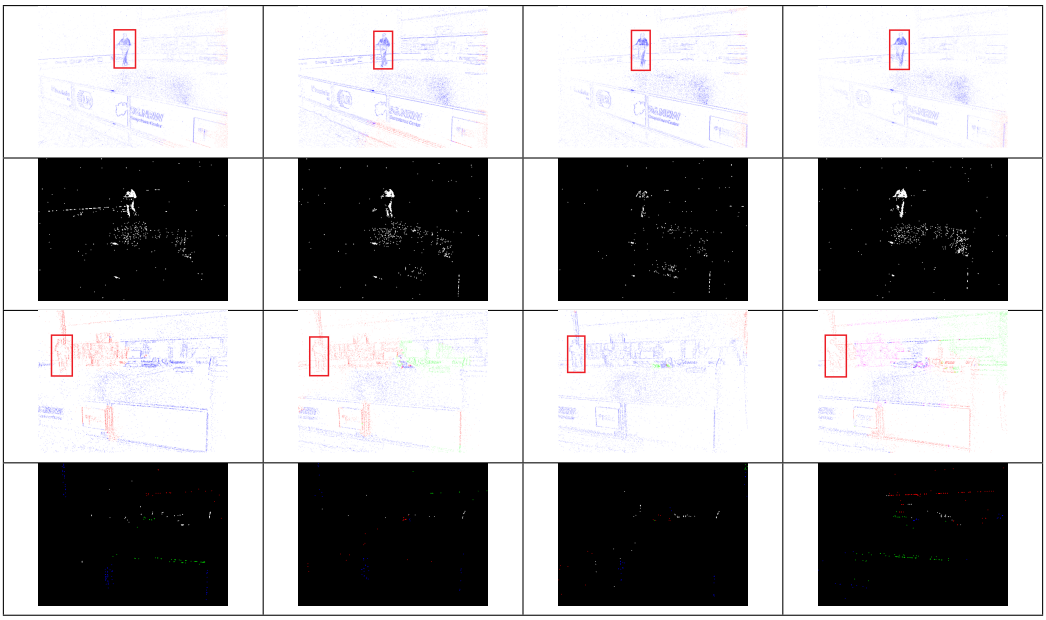


Table 6.2: Segmentation results and visualization of event data on the Custom Dataset

6.1.4 Custom Dataset

After evaluating the model's performance on various published datasets, the investigation was extended by applying the same methodology to a custom dataset. First, we tested the model with a scenario "slow_cam", which has a slow movement of the person with an object, while the camera moves away. The minimization parameters λ_D = 2, λ_P = 25, λ_M = 8,000 and $num\_events\_per\_image$ = 25,000, were selected after testing out various combinations, which gave satisfactory results for this scene, as shown in Table 6.2 (first row). Also, the hierarchical sub-divisions of the scene were tuned with the parameters $BracketRow$ = 6 and $BracketCol$ = 6. However, the model fails to segregate objects with different velocities, due to which it doesn't identify the IMOs. The event data output in the form of the tuple was later visualized, which showed the moving object in the scene, with some detections of the background, as given in Table 6.2 (second row).

The next experiment, titled "fast_cam," involved capturing a scene with rapid camera motion while a human subject moved within the frame. This scenario was designed to assess the model's robustness and accuracy under fast-motion conditions. Key model parameters were adjusted to optimize performance for this scene: λ_D was set to 2, λ_P to 22, λ_M to 10,000, with $num\_events\_per\_image$ increased to 30,000. Additionally, $BracketRow$ and $BracketCol$ were both set to 7, considering the object was positioned farther from the camera. The results, presented in Table 6.2 (third and fourth rows), show that the model struggled to distinguish between the motion of the object and the fast-moving background. The motion of the camera caused the model to cluster both the object's and background's movements into the same region, resulting in less effective motion segmentation. Furthermore, the event data produced by the model did not successfully isolate the moving object from its surroundings, highlighting the challenge of maintaining segmentation quality under such high-speed conditions.

6.1.5 Result Analysis

The evaluation of the energy minimization model for event-based motion segmentation reveals several key observations regarding its performance and adaptability across different datasets. While the model shows clear strengths in specific scenarios, there are also unusual limitations, particularly in fast and dynamic environments.

In well-controlled settings, such as the DVSMOTION20 and EED datasets, the model performs adequately. It demonstrated an ability to successfully isolate and segment motion patterns with a high degree of accuracy. For instance, in the "fast_drone" scenario, the model distinctly separated the motion of the drone from the background, which was largely affected by the camera's movement. This result illustrates the model's ability to differentiate fast-moving objects based on their velocities, indicating that, when parameterized correctly, it is highly effective for tasks requiring precise tracking and segmentation, where the camera's motion is quite slow compared to the object's motion.

Also, the aspect of segmentation in complex environments was visible, during the evaluation of datasets like EV-IMO, where multiple moving objects and camera motion needed to be segmented simultaneously. The model successfully differentiated the motions of two objects and captured the background dynamics induced by camera ego-motion. This suggests that the model performs well in multi-object environments with moderate scene complexity, provided that the velocity differences between objects are clear and the camera motion is relatively stable.

Despite the strengths, the model struggles in high-speed and highly dynamic scenes. For example, in the "fast_cam" scenario, where both the object and camera were in rapid motion, the model was unable to accurately separate the motion of the moving human subject from the fast-moving background. This led to poor segmentation results, as the model clustered both object and background movement into the same region. Additionally, over-segmentation was observed in multiple scenarios, where different regions of a single object were mistakenly classified as distinct motions. This suggests that the model tends to overfit the scene's motion patterns when the regularization parameters are not optimally set.

The model's performance is highly sensitive to several key parameters, which must be carefully tuned to achieve optimal results, as described in Table 6.3:

Parameter	Effect on Model Performance
$num\_events\_per\_image$	Increasing this value captures more motion patterns, improving segmentation accuracy in dynamic scenes, but risks over-segmentation or misclassification
λ_P	Controls the balance between over-segmentation and under-segmentation; increasing improves segmentation in complex or dynamic environments
λ_M	Ensures an appropriate number of motion models are applied; too many leads to overfitting or excessive clustering
λ_D	Directly affects the fitting of the motion models to data
$BracketRow\ and\ BracketCol$	Increasing these values captures finer motion details but can lead to segmentation errors if overused

Table 6.3: Key Parameters and their Effects on the Model's Performance

For scenes with high-speed dynamics, increasing these values can help capture finer motion details but may also introduce more segmentation errors if not balanced with other parameters. While the model demonstrates flexibility in adapting to various environments, this adaptability also adds complexity when deploying it in real-

world applications. In such scenarios, the need for manual tuning of parameters can be a limitation, making automatic parameter adjustment a more desirable feature for ensuring efficiency and ease of use.

One of the significant challenges identified in the analysis is the model's processing time for small datasets, typically containing a few seconds of event data. Depending on the specifications of the hardware being used, the model can take anywhere from several minutes to a few hours to produce results. This latency could limit its practical application, especially in time-sensitive scenarios or when operating with lower-end hardware configurations.

The energy minimization model [48], demonstrates strong segmentation capabilities in controlled environments, successfully isolating distinct motion patterns and handling multi-object scenarios with moderate complexity. However, its performance declines in fast, dynamic scenes, where it struggles to differentiate foreground objects from background motion and tends to over-segment. The model's effectiveness is highly dependent on careful parameter tuning, such as adjusting event data and regularization terms, which makes its deployment challenging in real-time applications. Additionally, its processing time limits its suitability for real-time applications.

6.2 Neural Network Approach for Depth and Motion Estimation

This section presents the experiments using the three networks outlined in Chapter 4. Each network was trained and evaluated on the EV-IMO dataset with several key hyperparameters, including learn_rate, epochs and batch_size.

Additionally, the growth_rate and final_map_size were fine-tuned to further enhance the model's performance. The growth_rate, which dictates how the number of feature maps expands with each layer, was carefully adjusted to ensure an optimal flow of information through the network. Meanwhile, the final_map_size, which determines the spatial dimensions of the output feature maps, was used to refine the desired level of detail in the final representations.

These values were initially based on those reported in [25], which provided a useful reference for configuring the set of encoder-decoder networks. The outputs of the models are visualized according to the colour scheme discussed in subsection 4.2.3.4, ensuring consistency in interpreting depth and motion estimation results. To streamline the process, a subset of the dataset was chosen for training, as it was deemed optimal for this type of network architecture [47]. From the six scenarios outlined in Table 6.4, one specific scenario was selected as the starting point, and the model was trained on that scene.

	background	speed	texture	occlusions	objects	light
$Set1$	boxes	low	medium	low	1-2	normal
$Set2$	floor/wall	low	low	low	1-3	normal
$Set3$	table	high	high	medium	2-3	normal
$Set4$	tabletop	low	high	high	1	normal
$Set5$	tabletop	medium	high	high	2	normal
$Set6$	boxes	high	medium	low	1-3	dark / flicker

Table 6.4: Categories of background geometry present in the EV-IMO dataset [25]

6.2.1 Evenly Cascaded Network (ECN)

To assess the performance of the ECN and to reproduce the results from [47, 25], we conducted a series of experiments using a variety of parameter configurations, including those outlined in the original study. A few of the outcomes resulting from the key hyperparameters are detailed in Table 6.5.

Sr. No	learn_rate	epochs	batch_size	growth_rate	final_map_size
1	5e-05	100	15	8	4
2	1e-03	30	32	4	8
3	1e-05	500	32	4	8

Table 6.5: Overview of the hyperparameter used in ECN experiments, with varying results across configurations

Configuration 1:

This configuration employs a balanced strategy, characterized by a moderate learning rate and a relatively small batch size. With a growth rate of 8 and a final feature map size of 4, the network demonstrates an effective feature extraction process, leading to strong segmentation performance in both depth and pose estimation tasks, as illustrated in Fig. 6.1a.

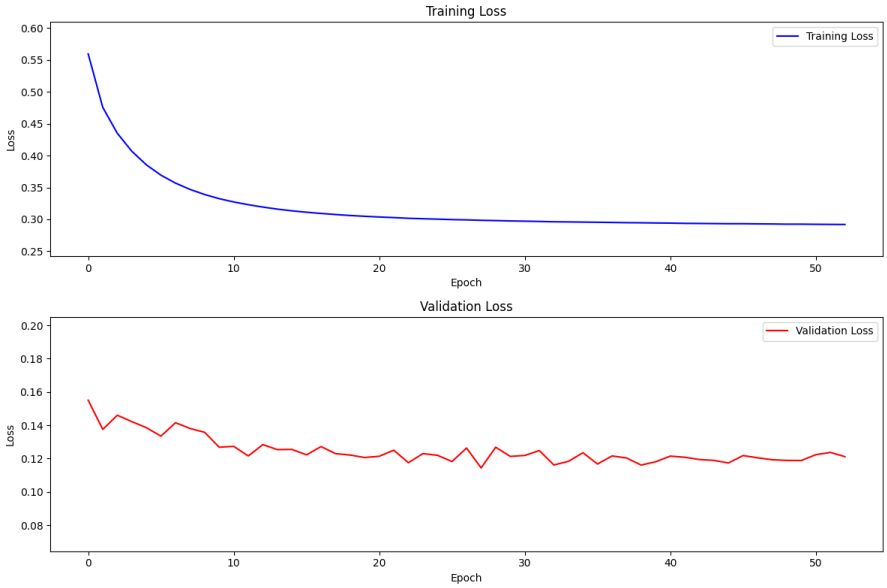
The first image displays the input test image, which serves as the basis for the subsequent analysis. Following this, the combined network output is presented, showcasing the estimated depth derived from the test image. Next, the optical flow visualization is illustrated, where a blue hue indicates that objects are moving to the left.

Accompanying these outputs, the ground truth depth mask is provided, which was utilized during the training phase alongside the input image. This mask serves as a benchmark for evaluating the accuracy of the depth estimations. The subsequent image displays the depth network's output, which reflects the model's performance in estimating depth for the test image, allowing for a comparative analysis against the ground truth. Finally, the last image illustrates the pixel-wise output from the pose network, capturing the predicted motion across the scene.

The inversion of depth output colours could potentially be attributed to illumination and shadow effects, where shadows are interpreted as being farther away, resulting in darker values within brighter areas. While individual depth estimations tend to accurately represent proximity, combining these outputs with the pose model may disrupt this representation, leading to a large block of white for the test object, with black inside, which is contrary to the ideal output where black signifies closeness.

As shown in Fig. 6.1b, the model achieved a final training loss of approximately 0.25, while the validation loss fluctuated before stabilizing around 0.13. This indicates that the model successfully converged during training and demonstrates better generalization to unseen data.

(a) Visualization of the models' inputs & outputs: The first image shows the input test image, followed by the combined network output for estimated depth and optical flow. The second image displays the ground truth depth mask used in training, the depth network's output for the test image, and the pixel-wise pose network's predicted motion output. The bounding boxes highlight the test object



(b) Training and validation loss over epochs

Figure 6.1: ECN Configuration 1: Summary of the network inputs, outputs, and training performance

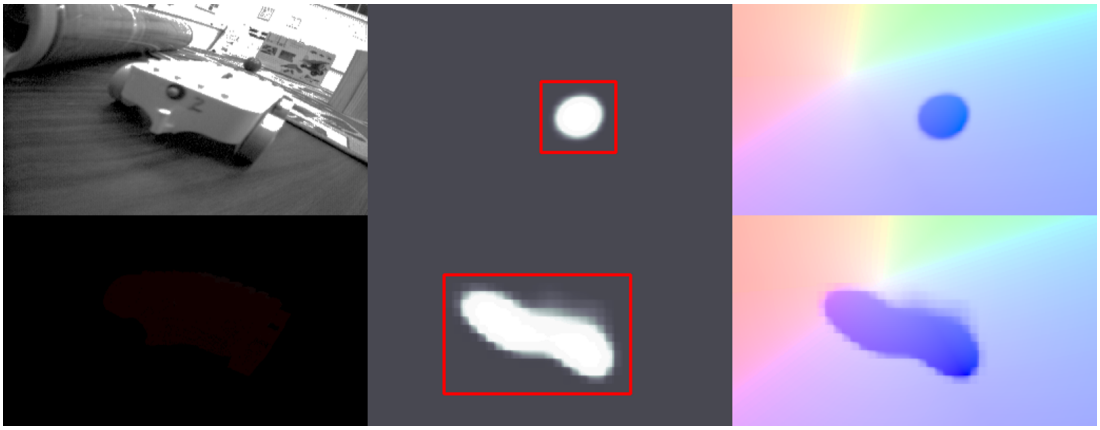
Configuration 2:

This configuration employs a more aggressive learning rate and a larger batch size, with a growth rate of 4 and a final feature map size of 8. These settings indicate a more streamlined feature extraction process. Despite the increased learning rate, the model shows moderate performance in both depth and pose estimation tasks, which can be attributed to the shorter training duration of 30 epochs.

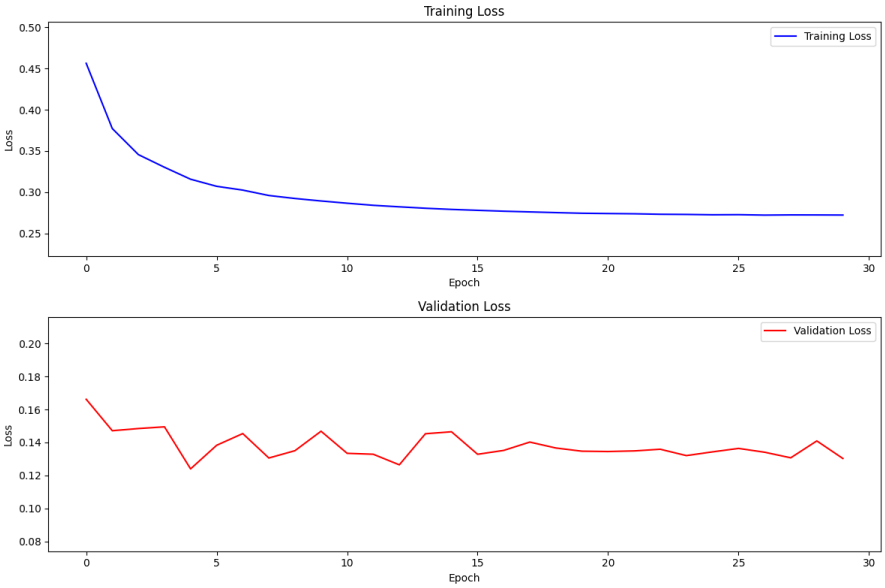
The network's depth estimation yielded less precise segmentation, with blurred boundaries and a lack of sharpness in the individual depth output. Regarding motion estimation, the pixel-wise pose network predicted the object's motion predominantly in blue, indicating movement in a left direction. However, the background exhibited a more complex motion pattern, characterized by a mixture of red, green, and blue colours, suggesting varying levels of motion across different regions. The combined network output, as well as the ego flow representation, showed a more abstract result, with a circular blob rather than distinct features, as illustrated in Fig 6.2a. This suggests that the model struggled to maintain the same level of detail in its predictions.

Fig 6.2b further provides insights into the model's performance throughout the training process. The training loss starts at a higher value and decreases significantly within the initial epochs, ultimately stabilizing around

0.27 by the end of the training. This decline indicates that the model was able to learn effectively from the training data, achieving a certain level of convergence despite the less sharp segmentation observed in the outputs. While, the validation loss starts at a comparable level to the training loss but stabilizes around 0.20, indicating that while the model is learning from the training data, its performance on unseen data does not significantly improve beyond a certain point.



(a) Visualization of the models' inputs & outputs: The depth estimation is outputted as a circular blob for the combined model, as compared to the individual network's output



(b) Training and validation loss over epochs

Figure 6.2: ECN Configuration 2: Summary of the network inputs, outputs, and training performance

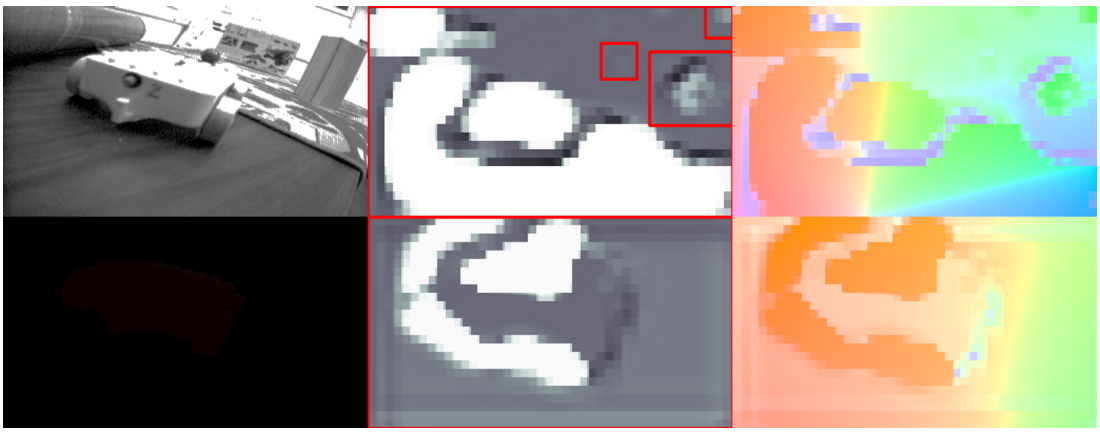
Configuration 3:

This configuration employs a learning rate of 1e-05, a batch size of 32, and a feature growth rate of 4, with a final feature map size of 8. The model was trained for 500 epochs, providing ample time for learning. However, despite the extended training period, the performance in both depth and pose estimation remained sub-optimal.

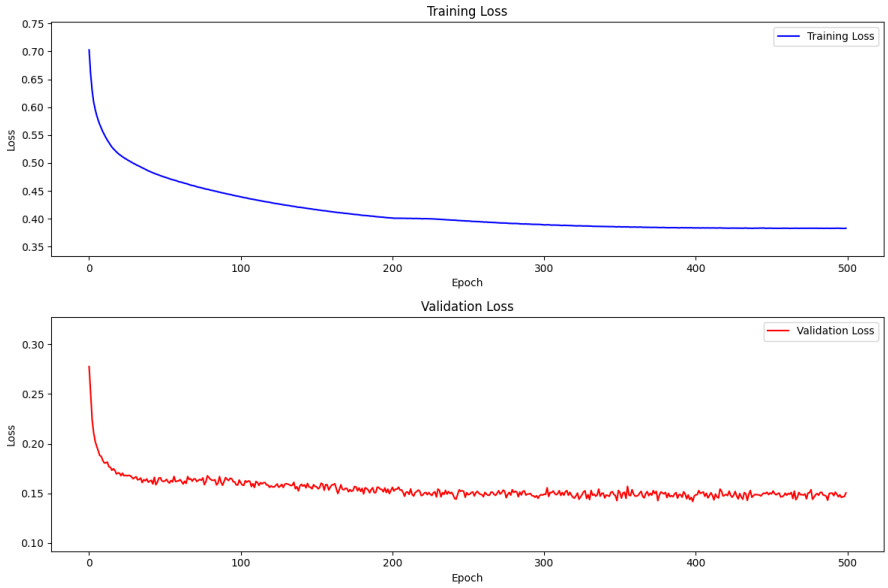
The depth estimation from the combined model output was heavily affected by varying lighting conditions, leading to distorted results. These outputs were dominated by white and grey patches, which obscured meaningful scene information, such as object boundaries. Similarly, the optical flow results were disrupted by inconsistent colour patterns, displaying multiple colours within the image, making it challenging to interpret object motion, as illustrated in Fig. 6.3a.

The individual depth network produced outputs with more observable object shapes compared to the com-

bined model. However, these shapes were still distorted and lacked the sharpness and precision necessary for accurate segmentation. The pixel-wise pose network also exhibited substantial distortion in its motion predictions, with notable artefacts in the output. These issues made it difficult to draw clear conclusions about object movement.



(a) Visualization of the models' inputs & outputs: The combined model output shows distortion with obscured object boundaries, while individual depth networks reveal more discernible shapes but lack precision. The pixel-wise pose network also demonstrates significant motion prediction artefacts, complicating the interpretation of object movement



(b) Training and validation loss over epochs

Figure 6.3: ECN Configuration 3: Summary of the network inputs, outputs, and training performance

As depicted in Fig. 6.3b, the training loss followed a consistent downward trend over the course of 500 epochs, starting at a relatively high value of 0.71 and gradually decreasing to 0.39. The validation loss exhibited a similar pattern, initially high before stabilizing around a plateau at 0.16. Despite this steady progress, the model's performance remained hindered by persistent challenges, particularly the inconsistent lighting in the dataset, which impaired its ability to produce high-quality depth and motion estimations in both the combined and individual network outputs.

6.2.2 Encoder-Decoder Network (EDN)

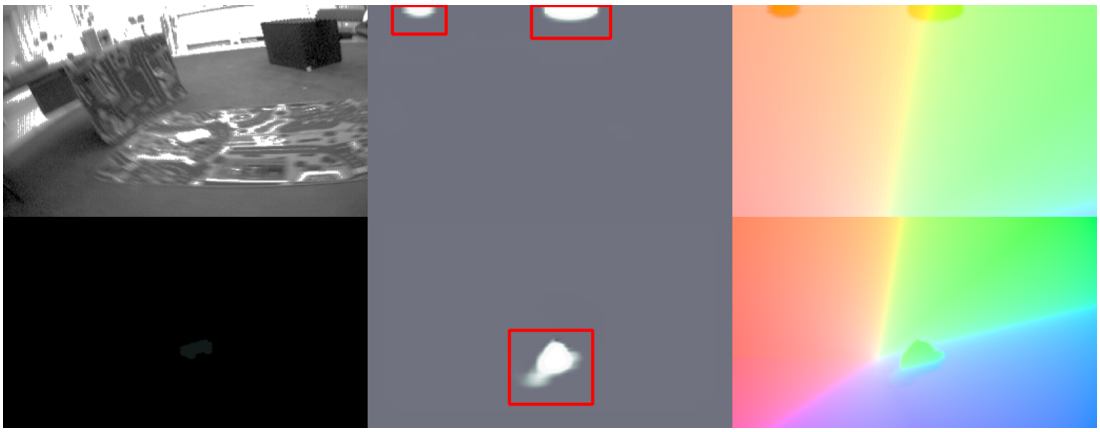
Simultaneously, the EDN was evaluated on the same dataset. Table 6.6 provides an overview of several configurations that were tested on the network, which gave mixed outcomes.

Sr. No	learn_rate	epochs	batch_size	growth_rate	final_map_size
1	5e-04	30	32	4	8
2	5e-05	30	15	8	4
3	1e-05	30	20	8	4

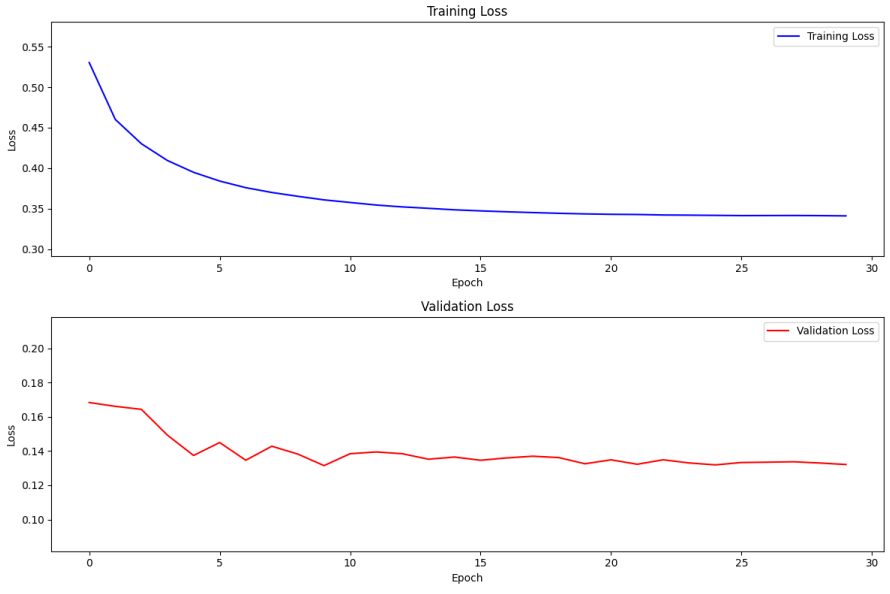
Table 6.6: Overview of the hyperparameter used in EDN experiments, with varying results across configurations

Configuration 1:

In this configuration, the model uses a learning rate of 5e-04, a batch size equal to 32, and a feature growth rate of 4, with a final feature map size of 8. The two sub-models were trained for 30 epochs. the depth and pose estimation tasks struggled to deliver strong results, likely due to the shorter training period and some complexities in the dataset.



(a) Visualization of the models' inputs & outputs: The combined model misidentified a background element, resulting in inaccurate depth and ego flow estimations. In contrast, the individual depth network provided clearer segmentation and optical flow predictions



(b) Training and validation loss over epochs

Figure 6.4: EDN Configuration 1: Summary of the network inputs, outputs, and training performance

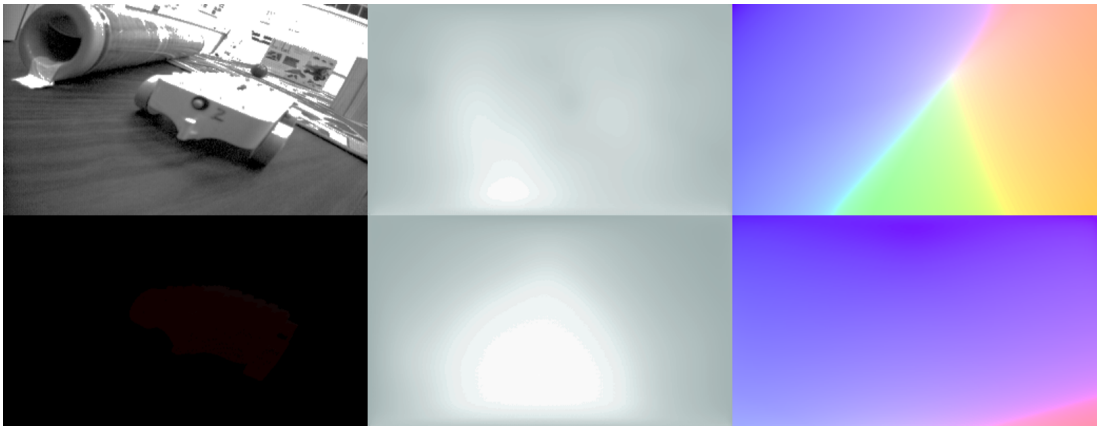
The combined model exhibited difficulties in detecting the target object, instead erroneously focusing on a background element in the test image. This misidentification resulted in incorrect ego flow and depth estima-

tion, suggesting that the fusion model was unable to reliably distinguish the object from its surroundings, as demonstrated in Fig. 6.4a. In contrast, the individual depth network performed with greater accuracy, producing a clearer segmentation of the target object, although the output remained somewhat imprecise. The optical flow predictions were also inconsistent—the object's motion was highlighted in green. Still, the background presented ambiguous motion patterns, indicating a lack of confidence in the model's predictions, as seen in the figure.

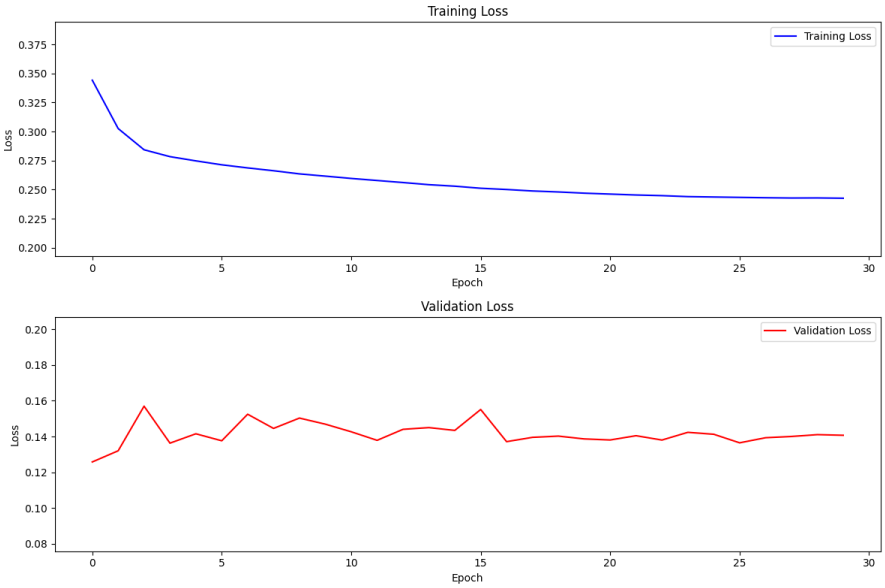
As shown in Fig. 6.4b, the training loss stabilizes around 0.35, with the validation loss decreasing to 0.13. However, despite these signs of convergence, the model fails to produce clear and accurate segmentation results.

Configuration 2:

The model parameters were adjusted as specified in Table 6.6. However, this configuration proved to be ineffective, as the model failed to produce meaningful estimations or the desired output, despite achieving lower training losses. Figs. 6.5a and 6.5b illustrate the model's performance during training and its blank outputs. This outcome highlights how changes in certain parameters can significantly affect the model's performance and results.



(a) Visualization of the models' inputs & outputs: No clear predictions for this configuration



(b) Training and validation loss over epochs

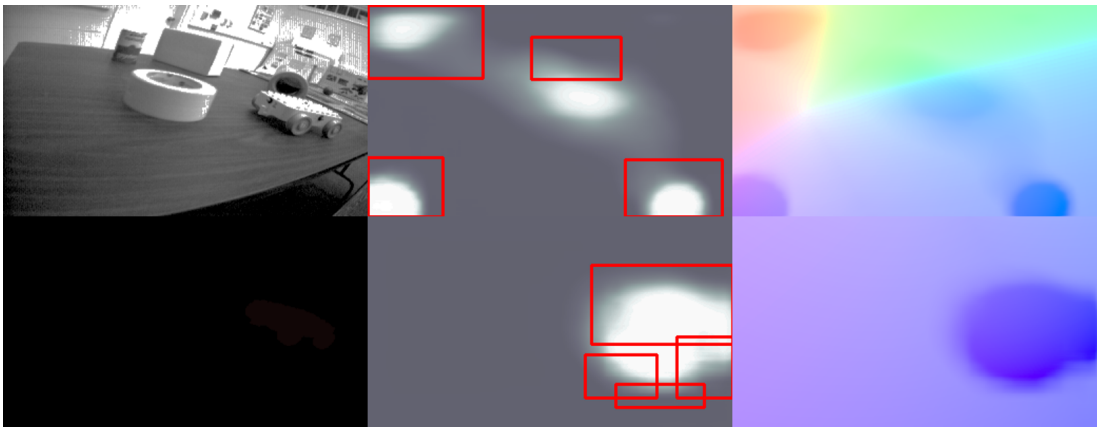
Figure 6.5: EDN Configuration 2: Summary of the network inputs, outputs, and training performance

Configuration 3:

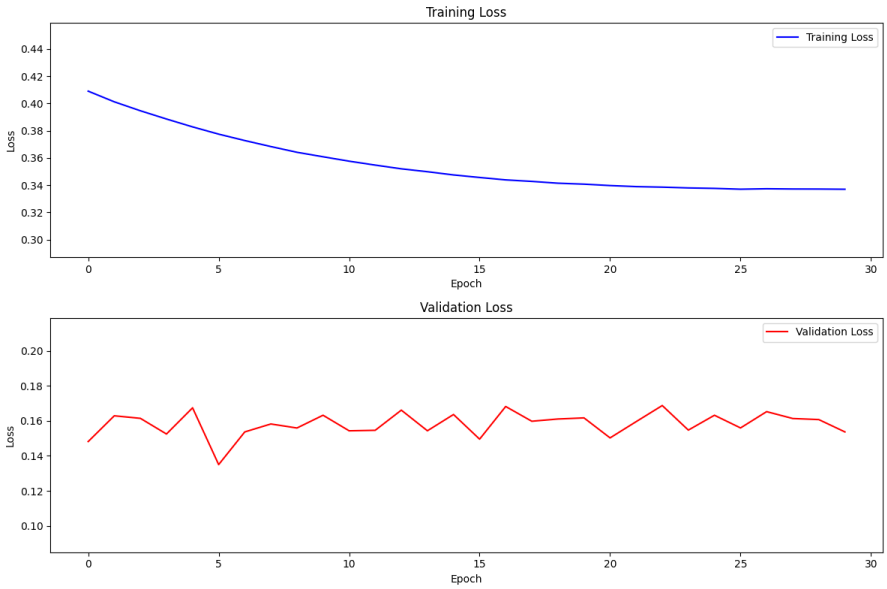
In this configuration, we utilized a learning rate of 1e-05, a batch size of 20, and 30 epochs, along with a growth rate of 8 and a final map size of 4. This setup aimed to balance the complexity of the model with the available data to improve performance on the depth and motion estimation tasks.

Fig. 6.6a illustrates the outcomes of the model. The combined output from the network exhibits noise detection manifesting as blurred white patches, which detracts from the clarity of the depth estimation and the ego-motion flow. Despite the issues, the individual model outputs show more promising results. The depth estimation for the test object appears cloudy, suggesting that while the model captures the general shape and position, it struggles to provide precise depth values. On the other hand, the motion flow output is characterized by a saturated blue hue for the object, indicating a stronger motion signal, while the background motion is represented in lighter shades of blue. This contrast effectively highlights the object's movement against a relatively static background, illustrating the model's capability to distinguish between varying motion dynamics.

The training loss is approximately 0.23, while the validation loss stabilizes at 0.12, indicating that the model fits the training set reasonably well. However, this fitting is not fully reflected in the network's output.



(a) Visualization of the models' inputs & outputs: The combined output shows blurred white patches affecting depth estimation and ego-motion flow, while individual model outputs display clearer depth estimation and enhanced motion flow, with saturated blue indicating stronger object motion



(b) Training and validation loss over epochs

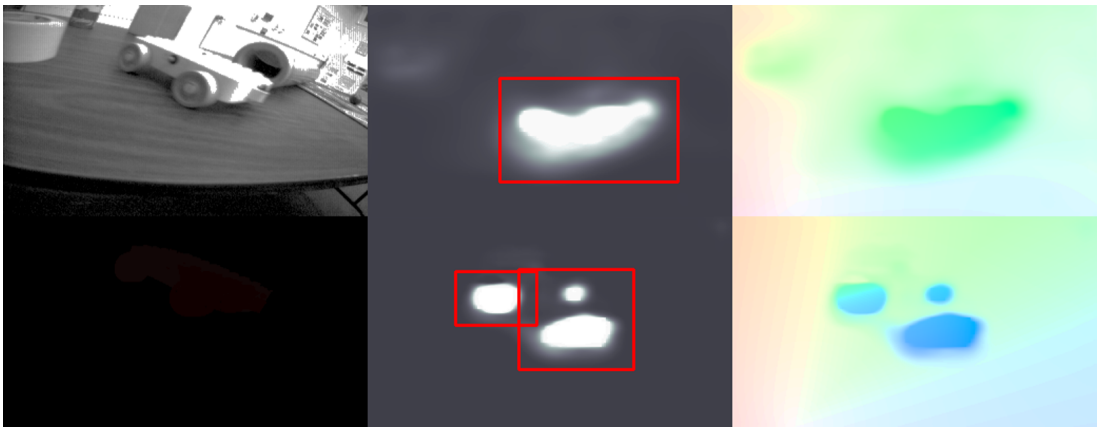
Figure 6.6: EDN Configuration 3: Summary of the network inputs, outputs, and training performance

6.2.3 ResNet18 Encoder-Decoder Network (REDN)

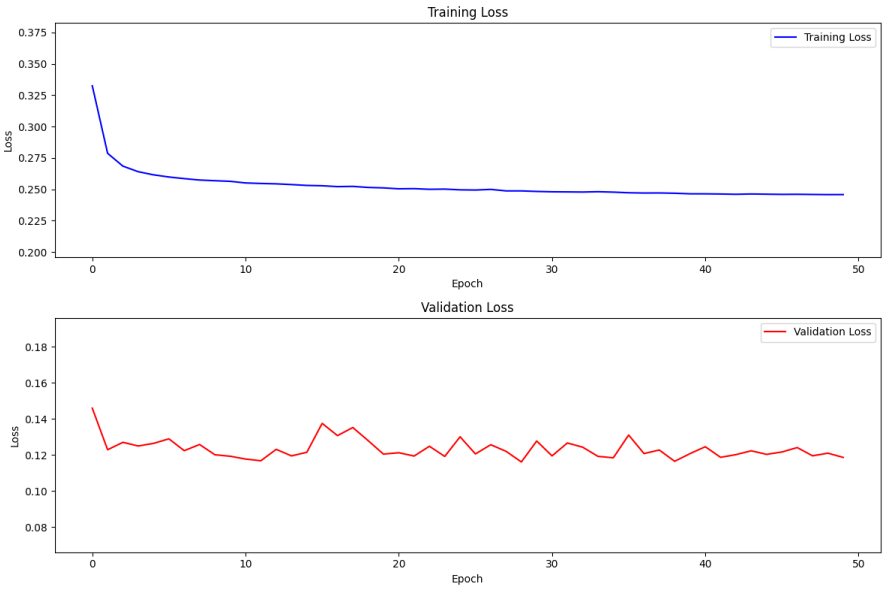
For the third set of evaluations, the REDN was assessed on the EV-IMO dataset, utilizing the feature extraction capabilities of ResNet18, as discussed in section 4.2. Due to limited computational resources, only a few configurations were tested, which resulted in average outcomes. While these configurations did not achieve exceptional results, they provided a reasonable foundation for understanding the model's performance within the constraints of the available resources.

Sr. No	learn_rate	epochs	batch_size	growth_rate	final_map_size
1	1e-02	50	32	4	8
2	1e-05	50	25	8	4

Table 6.7: Overview of the hyperparameter used in REDN experiments, with varying results across configurations



(a) Visualization of the models' inputs & outputs: The combined network output shows incoherent predictions, while individual networks partially detect the object. The per-pixel pose network indicates motion in blue for the object, with the background showing minimal movement



(b) Training and validation loss over epochs

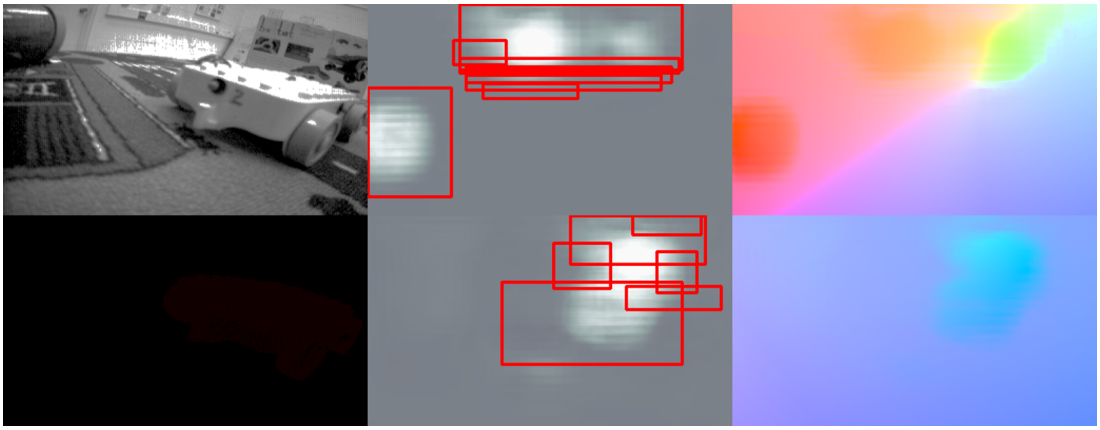
Figure 6.7: REDN Configuration 1: Summary of the network inputs, outputs, and training performance

Configuration 1:

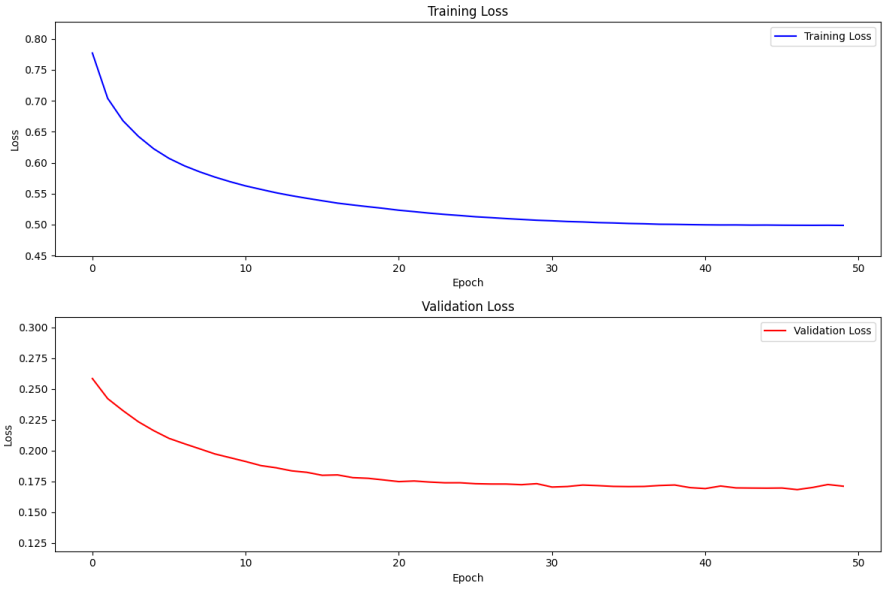
In this configuration, a relatively high learning rate of 1e-02 was used with a batch size of 32 and 50 epochs. The growth rate of 4 and a final map size of 8 aimed to balance the feature extraction depth and computational efficiency. The scale factor was set to 0.5, suggesting that the model focused on a broader spatial context for depth and motion estimation.

As per the outcome depicted in fig 6.7a, the combined network's output yielded random predictions of the test scene, including the corresponding optical flow, which lacked coherence. However, when examining the individual network's outputs, the object was partially detected, albeit inconsistently. The per-pixel pose network showed motion for the object in blue, while the background displayed no significant motion. These results suggest that, while the model was able to capture some movement details, the overall predictions were imprecise and struggled to provide a clear representation of the test scene.

The training loss exhibited a steep decline as training progressed, ultimately stabilizing at approximately 0.24. In contrast, the validation loss fluctuated significantly before settling at 0.12, as illustrated in Fig. 6.7b. Despite these relatively low loss values, the model's outputs remained inaccurate, with noisy depth estimations and inconsistent motion predictions. This suggests that, although the loss values indicate convergence, the model struggled to generalize effectively and produce reliable results.



(a) Visualization of the models' inputs & outputs: No clear estimations



(b) Training and validation loss over epochs

Figure 6.8: REDN Configuration 2: Summary of the network inputs, outputs, and training performance

Configuration 2: In this configuration, with a learning rate of 1e-05, a batch size of 25, and 50 epochs, the model struggled to deliver satisfactory results both in the combined output and individual predictions. The network failed to detect the object in the test scene, as illustrated in Fig. 6.8a. Additionally, the training and validation losses started at significantly high values and, although they eventually decreased, they settled at higher levels compared to other configurations. This indicates that the model had difficulty learning from the data and was unable to achieve optimal performance with these hyperparameters(Fig. 6.8b).

6.2.4 Result Analyis

In the evaluation of the three models (ECN, EDN, REDN) for the depth and pixel-wise pose estimation, a range of performance outcomes influenced primarily by the tuning of hyperparameters was observed, which provided varied outcomes, which are summarized in Table 6.8.

Various normalization methods were also tested during training, including group normalization, batch normalization, and feature decorrelation. Of these, Group Normalization was primarily used, yielding more stable training and consistent convergence, particularly when working with smaller batch sizes. Other normalization methods showed mixed results, occasionally causing fluctuations in training loss or less reliable outputs. Adjustments to channel sizes and the growth rate of the model had a noticeable impact on performance, particularly in terms of balancing the model's complexity with its computational efficiency.

The performance of each model was also affected by tuning the weights of the loss functions (as mentioned in 4.2.4, used during training. Careful balancing of these losses between depth and pose networks—resulted in improved accuracy and smoother convergence for some configurations. Zero-padding and border-padding were experimented with, primarily affecting the object boundaries in depth maps. While these techniques introduced subtle improvements, they were not as critical as the main hyperparameters in determining the quality of the model outputs.

Model Performance:

Model	Key Points	Results
ECN	Moderate learning rate, smaller batch size, growth rate of 8	The network produced the best results amongst the three, with strong convergence and sharp feature extraction. Configuration 1 performed the best individually, while Configuration 3 struggled with inconsistent lighting in the dataset.
EDN	Varied learning rates, batch sizes between 15-32, growth rate of 4 or 8	EDN delivered moderate results, with Configuration 1 yielding clearer object segmentation. However, lower learning rates and certain parameter tweaks in Configuration 2 led to failure in providing meaningful predictions.
REDN	High learning rate, batch size of 32	REDN performed the worst, with erratic outputs in both depth and motion estimations. Despite tuning efforts, it struggled to detect objects in test images and failed to generalize effectively.

Table 6.8: Model performance summary

In conclusion, while secondary factors such as normalization methods, channel sizes, and padding strategies influenced the results, hyperparameter tuning—especially of learning rates, batch sizes, and epochs—proved to be the most significant factor in model performance. ECN showed the strongest results overall, followed by EDN, with REDN under-performing across most configurations.

6.3 Experimental Summary

The evaluation of both the energy minimization model and the neural network-based approach for event-based motion segmentation highlights distinct strengths and limitations inherent to each methodology.

The energy minimization model exhibits strong capabilities in isolating and segmenting objects based on their velocities, particularly in controlled environments. It effectively generates distinct motion models for various objects, demonstrating robust performance in moderate and stable scenarios, such as those presented in the published datasets. However, the model encounters significant challenges in highly dynamic scenes, where both camera and object movements are rapid. In such cases, the model struggles to differentiate the objects in motion from the background or the other object's motion, resulting in issues such as over-segmentation and poor classification. Moreover, a substantial limitation of this approach is its computational efficiency; processing times are often lengthy, making the model unsuitable for real-time applications or situations that require scalability.

In contrast, the neural network-based approach offers a more adaptable and comparatively faster solution. Although its performance varies based on specific configurations, the models deployed and the quality of the dataset used, it consistently outperforms the energy minimization model in terms of speed and, in several instances, accuracy. The neural network models demonstrate greater flexibility in handling both depth and pose estimation efficiently. Furthermore, this approach is well-suited for real-time application, enhancing its scalability and practicality for real-world scenarios. Nonetheless, the performance of the neural network models remains contingent upon the quality of the training data, with improved datasets likely yielding even better results.

In summary, while the energy minimization model demonstrates strong performance in detailed motion segmentation within controlled environments, achieving an accuracy of 76.81%, the neural network-based approach slightly surpasses this with an accuracy of 77% [48]. However, the energy minimization model faces limitations due to its slow computation speed and difficulties in handling fast dynamic scenes, which hinder its practical application. In contrast, the neural network-based method offers a faster and more scalable solution, making it well-suited for real-time implementation, particularly as the quality of training datasets continues to improve.

6.4 Algorithms and Code

The implemented code and the models for the motion segmentation task based on neural networks can be accessed on GitHub repository **MSDEEC**: https://github.com/rahulpanchal7/MSDEEC

Access to the results and the implemented model is restricted to the members of TU Dortmund University.

7 Conclusion & Outlook

The primary aim of this study was to develop a robust approach capable of segmenting a moving object from its background or another object in dynamic scenes, even when the camera itself is in motion. To achieve this, an event camera was used, which captures only the changes in pixel intensity, thereby offering a unique advantage in dynamic environments. The study explored two established approaches, building on each to improve performance.

The first approach, based on energy minimization, produced promising results, particularly in managed settings. However, it came with significant computational demands, resulting in slow processing times, which are unsuitable for real-time applications. Additionally, the model required parameter tuning specific to each scene, reducing its scalability and limiting its capability to generalise to diverse environments. Despite these challenges, this method set a strong foundation for further exploration and provided a solid benchmark for comparison.

This prompted the transition to a neural network-based approach, where the same optical flow logic was implemented through deep learning. By incorporating depth masks along with the images, the models produced more accurate segmentations. Among the three different models tested, the ECN consistently delivered the best results, outperforming the others in terms of accuracy and reliability. The EDN, while yielding mixed results compared to those reported in the literature [25], showed potential but required more optimization. A variant of EDN, which used a ResNet18-based encoder i.e. REDN, was also tested to enhance feature extraction capabilities. However, this model did not perform well, likely due to lower-quality training data and the constraints imposed by high computational demands, which forced the model to be trained with sub-optimal hyperparameters.

Despite the obstacles encountered, the neural network-based approach emerged as a promising solution, particularly due to its adaptability and ability to handle event-based data efficiently. Its potential for real-time application and scalability makes it a more practical choice for dynamic scenes involving both moving objects and cameras. The comparative evaluation of the energy minimization and neural network models highlights distinct trade-offs. While the energy minimization approach performs well in controlled, stable environments, it faces challenges in highly dynamic scenes, particularly due to its computational inefficiency. In contrast, neural networks offer a more adaptable, scalable, and faster solution, making them better suited for real-world, real-time applications where both camera and object motion are involved.

In conclusion, this study primarily focused on achieving effective motion segmentation using event cameras, while also providing an indirect comparison between classical and data-driven approaches. The energy minimization method demonstrated effectiveness in specific scenarios but was hindered by slow processing speeds and limited adaptability, reducing its practical utility. In contrast, neural networks emerged as a more promising alternative, offering greater flexibility and faster processing, making them better suited for real-time applications.

Although the study yielded partial results, particularly regarding the application of the model directly to video data, it established a strong foundation for future research. With ongoing advancements in data quality and computational resources, neural networks are well-positioned to further improve the accuracy and applicability of motion segmentation in dynamic environments. Overall, this research highlights both the strengths and limitations of the classical and data-driven approaches, providing valuable insights for future developments in the field.

7.1 Future Scope

This thesis has highlighted the potential of event cameras for motion segmentation in dynamic environments, while also identifying several areas for further development. A key direction for future work is improving data quality. Leveraging more diverse, higher-resolution datasets—particularly those with detailed depth masks—could enhance segmentation and tracking accuracy across a broader range of dynamic scenes. High-quality data would also improve the model's ability to generalize to different environments. On the technical front, future research should focus on architectural innovation in convolutional neural networks (CNNs) or hybrid deep learning frameworks. By integrating more sophisticated models, such as spatio-temporal networks or attention-based mechanisms, the precision and efficiency of both object detection and motion segmentation could be substantially increased. Moreover, incorporating 3D motion models that capture the intricate spatial and temporal relationships within dynamic scenes would be particularly advantageous. This could pave the way for more effective multi-object tracking in environments where multiple entities and the camera itself are in simultaneous motion, addressing the complexities of occlusion and scale variation in real time. Another critical area for advancement is real-time performance. Optimizing the neural networks for real-time operation, possibly through hardware acceleration (such as GPUs) or edge computing, would make these models suitable for practical applications in fields like autonomous driving, embedded platforms or robotics. Additionally, combining depth estimation with robust tracking mechanisms could enable more reliable video tracking and consistent object detection over extended sequences. Looking ahead, expanding these models to support multi-object segmentation with precise pose and depth estimation could unlock new possibilities in 3D object reconstruction and comprehensive scene understanding. These advancements would significantly enhance the practical utility of event cameras in real-world dynamic environments, making them a powerful tool in various applications.

Bibliography

[1] Amit Aflalo et al. *DeepCut: Unsupervised Segmentation using Graph Neural Networks Clustering*. 2023. arXiv: 2212.05853 [cs.CV]. URL: https://arxiv.org/abs/2212.05853.

[2] Charu Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Jan. 2018. ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

[3] Himanshu Akolkar, Sio-Hoi Ieng, and Ryad Benosman. "See before you see: Real-time high speed motion prediction using fast aperture-robust event-driven visual flow". In: *CoRR* abs/1811.11135 (2018). arXiv: 1811.11135. URL: http://arxiv.org/abs/1811.11135.

[4] Yusra Alkendi et al. *Neuromorphic Vision-based Motion Segmentation with Graph Transformer Neural Network*. 2024. arXiv: 2404.10940.

[5] Mohammed Almatrafi et al. *Distance Surface for Event-Based Optical Flow*. 2020. arXiv: 2003.12680 [eess.IV]. URL: https://arxiv.org/abs/2003.12680.

[6] Ahmet Aytekin. "Comparative Analysis of the Normalization Techniques in the Context of MCDM Problems". In: 4 (Mar. 2021), pp. 1–25. DOI: 10.31181/dmame210402001a.

[7] Marc de Berg et al. *Computational Geometry: Algorithms and Applications*. 3rd. Berlin, Germany: Springer, 2009.

[8] Daniel Berrar. "Cross-Validation". In: Jan. 2018. ISBN: 9780128096338. DOI: 10.1016/B978-0-12-809633-8.20349-X.

[9] Houda Bichri, Adil Chergui, and Hain Mustapha. "Investigating the Impact of Train / Test Split Ratio on the Performance of Pre-Trained Models with Custom Datasets". In: *International Journal of Advanced Computer Science and Applications* 15 (Jan. 2024). DOI: 10.14569/IJACSA.2024.0150235.

[10] Pia Katalin Bideau. "Motion Segmentation - Segmentation of Independently Moving Objects in Video". Doctoral Dissertation. University of Massachusetts Amherst, 2020. DOI: 10.7275/w9vx-9171. URL: https://scholarworks.umass.edu/dissertations_2/1812.

[11] Giannis Chantas, Theodosios Gkamas, and Christophoros Nikou. "Variational-Bayes Optical Flow". In: *Journal of Mathematical Imaging and Vision* 50 (Nov. 2014). DOI: 10.1007/s10851-014-0494-3.

[12] Guillermo Gallego et al. "Event-Based Vision: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (2022), pp. 154–180. DOI: 10.1109/TPAMI.2020.3008413.

[13] Hossein Gholamalinezhad and Hossein Khosravi. "Pooling Methods in Deep Neural Networks, a Review". In: *CoRR* abs/2009.07485 (2020). arXiv: 2009.07485. URL: https://arxiv.org/abs/2009.07485.

[14] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[15] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[16] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[17] Berthold K.P. Horn and Brian G. Schunck. "Determining optical flow". In: *Artificial Intelligence* 17.1 (1981), pp. 185–203. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. URL: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[18] Luke Joel and Wesley Doorsamy. “A Review of Missing Data Handling Techniques for Machine Learning”. In: (Sept. 2022). DOI: 10.15157/IJITIS.2022.5.3.971-1005.

[19] Johannes Lederer. “Activation Functions in Artificial Neural Networks: A Systematic Overview”. In: *CoRR* abs/2101.09957 (2021). arXiv: 2101.09957. URL: https://arxiv.org/abs/2101.09957.

[20] Feng Liu and Jian-ya Gong. “Image Segmentation Using Energy Minimization and Markov Random Fields”. In: *2011 International Conference on Computer and Management (CAMAN)*. 2011, pp. 1–4. DOI: 10.1109/CAMAN.2011.5778751.

[21] Xinghua Liu et al. “Event Camera-based Motion Segmentation via Depth Estimation and 3D Motion Compensation”. In: *2022 41st Chinese Control Conference (CCC)*. 2022, pp. 6742–6747. DOI: 10.23919/CCC55666.2022.9902710.

[22] Ana I. Maqueda et al. “Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars”. In: *CoRR* abs/1804.01310 (2018). arXiv: 1804.01310. URL: http://arxiv.org/abs/1804.01310.

[23] Jana Mattheus, Hans Grobler, and Adnan M. Abu-Mahfouz. “A Review of Motion Segmentation: Approaches and Major Challenges”. In: *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. 2020, pp. 1–8. DOI: 10.1109/IMITEC50163.2020.9334076.

[24] Nico Messikommer et al. *Data-driven Feature Tracking for Event Cameras*. 2023. arXiv: 2211.12826 [cs.CV]. URL: https://arxiv.org/abs/2211.12826.

[25] Anton Mitrokhin et al. *EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras*. 2019. arXiv: 1903.07520 [cs.CV].

[26] Anton Mitrokhin et al. “Event-based Moving Object Detection and Tracking”. In: *CoRR* abs/1803.04523 (2018). arXiv: 1803.04523. URL: http://arxiv.org/abs/1803.04523.

[27] Anton Mitrokhin et al. “Event-based Moving Object Detection and Tracking”. In: *CoRR* abs/1803.04523 (2018). arXiv: 1803.04523. URL: http://arxiv.org/abs/1803.04523.

[28] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. “Fast Event-based Corner Detection”. In: Sept. 2017. DOI: 10.5244/C.31.33.

[29] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[30] OpenAI. *ChatGPT (3.5)*. 2024. URL: https://chat.openai.com.

[31] Federico Paredes-Vallés, Kirk Y. W. Scheper, and Guido C. H. E. de Croon. “Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception”. In: *CoRR* abs/1807.10936 (2018). arXiv: 1807.10936. URL: http://arxiv.org/abs/1807.10936.

[32] Daniel Peña and Víctor J. Yohai. “A Review of Outlier Detection and Robust Estimation Methods for High Dimensional Time Series Data”. In: *Econometrics and Statistics* (2023). ISSN: 2452-3062. DOI: https://doi.org/10.1016/j.ecosta.2023.02.001. URL: https://www.sciencedirect.com/science/article/pii/S2452306223000084.

[33] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. “ESIM: an Open Event Camera Simulator”. In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 969–982. URL: https://proceedings.mlr.press/v87/rebecq18a.html.

[34] Sylvestre-Alvise Rebuffi et al. “Data Augmentation Can Improve Robustness”. In: *CoRR* abs/2111.05328 (2021). arXiv: 2111.05328. URL: https://arxiv.org/abs/2111.05328.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[36] Laura Sevilla-Lara et al. “Optical Flow with Semantic Segmentation and Localized Layers”. In: *CoRR* abs/1603.03911 (2016). arXiv: 1603.03911. URL: http://arxiv.org/abs/1603.03911.

[37] C. Stauffer and W.E.L. Grimson. “Adaptive background mixture models for real-time tracking”. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 2. 1999, 246–252 Vol. 2. DOI: 10.1109/CVPR.1999.784637.

[38] Timo Stoffregen et al. “Event-Based Motion Segmentation by Motion Compensation”. In: *"Int. Conf. Comput. Vis. (ICCV)"*. 2019.

[39] Timo Stoffregen et al. “Event-Based Motion Segmentation by Motion Compensation”. In: *CoRR* abs/1904.01293 (2019). arXiv: 1904.01293. URL: http://arxiv.org/abs/1904.01293.

[40] Jasjit Suri et al. “UNet Deep Learning Architecture for Segmentation of Vascular and Non-Vascular Images: A Microscopic Look at UNet Components Buffered With Pruning, Explainable Artificial Intelligence, and Bias”. In: *IEEE Access* PP (Jan. 2022), pp. 1–1. DOI: 10.1109/ACCESS.2022.3232561.

[41] Kazuki Suzuki. *Minkowski spacetime diagram*. https://bingweb.binghamton.edu/~suzuki/ModernPhysics/2_Minkowski_spacetime_diagram.pdf. Accessed: 2024-09-22.

[42] S. M. Nadim Uddin, Soikat Hasan Ahmed, and Yong Ju Jung. “Unsupervised Deep Event Stereo for Depth Estimation”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.11 (2022), pp. 7489–7504. DOI: 10.1109/TCSVT.2022.3189480.

[43] S. Vani, T. V. Madhusudhana Rao, and Ch. Kannam Naidu. “Comparative Analysis on variants of Neural Networks: An Experimental Study”. In: *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. 2019, pp. 429–434. DOI: 10.1109/ICACCS.2019.8728327.

[44] Johan Vertens, Abhinav Valada, and Wolfram Burgard. “SMSnet: Semantic motion segmentation using deep convolutional neural networks”. In: Sept. 2017, pp. 582–589. DOI: 10.1109/IROS.2017.8202211.

[45] Mingle Xu et al. “A Comprehensive Survey of Image Augmentation Techniques for Deep Learning”. In: *Pattern Recognition* 137 (2023), p. 109347. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2023.109347. URL: https://www.sciencedirect.com/science/article/pii/S0031320323000481.

[46] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (2020), pp. 295–316. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.07.061. URL: https://www.sciencedirect.com/science/article/pii/S0925231220311693.

[47] Chengxi Ye et al. “Unsupervised Learning of Dense Optical Flow and Depth from Sparse Event Data”. In: *CoRR* abs/1809.08625 (2018). arXiv: 1809.08625. URL: http://arxiv.org/abs/1809.08625.

[48] Yi Zhou et al. “Event-based Motion Segmentation with Spatio-Temporal Graph Cuts”. In: *IEEE Transactions on Neural Network and Learning Systems* (2021).

[49] Alex Zhu et al. “EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras”. In: *Robotics: Science and Systems XIV*. RSS2018. Robotics: Science and Systems Foundation, June 2018. DOI: 10.15607/rss.2018.xiv.062. URL: http://dx.doi.org/10.15607/RSS.2018.XIV.062.

List of Figures

1.1 DVXplorer Camera for capturing event data [33] . . . 3

2.1 Comparison of a standard and an event camera's viewing [33] . . . 5

2.2 Feature detection and tracking using event camera data, highlighting the process of extracting significant features for a frame attention network, which ensures robust tracking across dynamic scenes [24]. . . . 7

2.3 Estimation of optical flow using the EV-FlowNet framework, which adeptly captures motion dynamics in event-based data, enhancing both accuracy and performance [49]. . . . 7

2.4 The reconstructed scene displays a semi-dense map coloured by depth, overlaid with events in grey, demonstrating excellent alignment between the map and events. The estimated camera trajectory from various methods and the semi-dense 3D map (point cloud) are also depicted [12] 8

2.5 Segmentation results on the EVIMO dataset: Gray is used for the background and dark red/blue for moving objects [48] . . . 9

2.6 ct denotes the time axis and x represents the spatial axis (a line). The diagram is a plane, with light rays traveling at 45° and 135° angles to the spatial axis (these rays form a two-dimensional light cone). The blue line represents the world line for accelerated motion, while the red line represents the world line of constant motion [41]. . . . 11

2.7 Delauanay triangulation, with circumcircles [7] . . . 11

2.8 Undirected graph, as a joint distribution using MRF [20] . . . 12

2.9 Neuron in a neural network [2] . . . 14

2.10 Multi-layer neural network [43] . . . 14

2.11 Linear Activation Function [19] . . . 16

2.12 Non-Linear Activation Function [19] . . . 16

2.13 Sigmoid/Logistic Function [19] . . . 17

2.14 Tanh Function [19] . . . 17

2.15 ReLU Function [19] . . . 18

2.16 Single Layered Perceptron [2] . . . 18

2.17 Recurrent Neural Network [43] . . . 19

2.18 CNN Architecture [43] . . . 20

2.19 A visual representation of a convolutional layer: the central element of the kernel is positioned over the input vector, then computed and replaced with a weighted sum of itself and the surrounding pixels [29] 21

2.20 Max Pooling[29] 22

2.21 Representation of the U-Net Architecture [35] 23

2.22 Types of Residual block [14, 15] 25

2.23 Overview of the ResNet architecture [15] 25

4.1 Flowchart of the motion segmentation process using event-based cameras [48] 28

4.2 Construction of the spatio-temporal graph [48] 29

4.3 A depth network is trained with supervision to predict scene depth. Meanwhile, a pose network processes sequential event slices to create a pixel-wise pose mixture model, yielding both pose mixtures and their associated probabilities. By combining the outputs from these two networks, optical flow is generated [25]. 32

4.4 **Evenly Cascaded Network**: This diagram illustrates the working principle of the cascading network. The network employs a multi-level feature extraction and refinement process. Features are extracted at different levels of abstraction, and feedback signals are incorporated to improve feature quality. The final prediction is generated through a series of refinement stages, combining features from multiple levels. [47]. 33

4.5 **DispNetS Architecture**: The network's encoder down-samples the input through seven convolutional layers, increasing feature channels. The decoder up-samples using transposed convolutions with skip connections. Final disparity maps are used for depth estimation [25] 35

4.6 **PoseExpNet Architecture**: The network processes stacked images through convolutional layers to predict 6-DoF camera pose and generates an explainability mask to highlight unreliable regions. Optional disparity maps can also be predicted [25] 36

4.7 **ResDepthNet Architecture**: The network uses a pre-trained ResNet18 as the encoder, followed by a decoder that up-samples feature maps using transposed convolutions. Disparity maps are predicted at multiple scales and refined with skip connections for depth estimation 37

4.8 **ResPoseNet Architecture**: Using ResNet18 as the encoder, ResPoseNet processes stacked images to predict 6-DoF camera pose and generate explainability masks. Optionally, disparity maps are predicted at multiple resolutions for depth estimation 38

6.1 ECN Configuration 1: Summary of the network inputs, outputs, and training performance 50

6.2 ECN Configuration 2: Summary of the network inputs, outputs, and training performance 51

6.3 ECN Configuration 3: Summary of the network inputs, outputs, and training performance 52

6.4 EDN Configuration 1: Summary of the network inputs, outputs, and training performance 53

6.5 EDN Configuration 2: Summary of the network inputs, outputs, and training performance 54

6.6 EDN Configuration 3: Summary of the network inputs, outputs, and training performance 55

6.7 REDN Configuration 1: Summary of the network inputs, outputs, and training performance . . . 56

6.8 REDN Configuration 2: Summary of the network inputs, outputs, and training performance . . . 57

List of Tables

4.1 Direction Representation in Optical Flow 39

4.2 Magnitude Representation in Optical Flow 39

6.1 Segmentation results on the DVSMOTION20, EED, and EV-IMO datasets. Time runs from left to right. The ground truth bounding box (in red) indicates the manually annotated 2D location of the IMOs. 45

6.2 Segmentation results and visualization of event data on the Custom Dataset 46

6.3 Key Parameters and their Effects on the Model's Performance 47

6.4 Categories of background geometry present in the EV-IMO dataset [25] 48

6.5 Overview of the hyperparameter used in ECN experiments, with varying results across configurations 49

6.6 Overview of the hyperparameter used in EDN experiments, with varying results across configurations 53

6.7 Overview of the hyperparameter used in REDN experiments, with varying results across configurations 56

6.8 Model performance summary 58

List of Abbreviations

DAVIS Dynamic and Active-pixel Vision Sensor
DVS Dynamic Vision Sensor
MRF Markov Random Field
IWE Inverse Warped Event
SNN Spiking Neural Networks
ReLU Rectified Linear Unit
ANN Artificial Neural Network
CNN Convolution Neural Network
GNN Graph Neural Networks
IWE Images of Warped Event
IMO Independent Moving Object
ECN Evenly Cascaded Network
EDN Encoder-Decoder Network
REDN ResNet18 Encoder-Decoder Network
MDL Minimum Description Length
EED Extreme Event Dataset

Eidesstattliche Versicherung

(Affidavit)

Panchal, Rahul Shashank — 236510

Name, Vorname (surname, first name) — Matrikelnummer (student ID number)

☐ Bachelorarbeit (Bachelor's thesis) ■ Masterarbeit (Master's thesis)

Titel (Title)

Motion Segmentation in Dynamic Environment using Event Cameras

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 09/10/2024

Ort, Datum (place, date)

Unterschrift (signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 09/10/2024

Ort, Datum (place, date)

Unterschrift (signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**