Learning Event Representations for Vision Foundation Models for Monocular Depth Estimation

Master Thesis

Maosheng Jiang

Delft University of Technology

TU Delft

Learning Event Representations for Vision Foundation Models for Monocular Depth Estimation

by

Maosheng Jiang

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday June 25, 2026 at 15:00.

Student number: 5586305
Project duration: November 10, 2025 – June 25, 2026
Thesis committee: H. Araghi, TU Delft, Daily Supervisor
N. Tömen, TU Delft, Supervisor
M. Weinmann, TU Delft, External Committee Member

TU Delft

Preface

This thesis marks the end of five years at Delft University of Technology. It has been a journey with challenges, setbacks, and many good moments. I am grateful for the opportunity to have worked with and learned from many people, and I hope this work contributes, even in a small way, to the field of event cameras.

Before starting my studies, I became interested in computer vision through seeing early applications of autonomous systems and camera-based sensing in everyday life. This motivated me to join the Data Science and AI programme, specialize in computer vision, and eventually work on depth estimation with event cameras.

I would like to thank my supervisors, H. Araghi and N. Tömen, for their guidance, support, and the knowledge they shared throughout this work. Their feedback and discussions were essential to this thesis.

Finally, I would like to thank my family and friends for their support during this period.

With this chapter coming to an end, I look forward to what comes next.

Maosheng Jiang
Delft, June 2026

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
Abs Rel	Absolute Relative Error
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Augmented Reality
AR/VR	Augmented Reality / Virtual Reality
CARLA	CARLA Driving Simulator
CNN	Convolutional Neural Network
DAE	Depth AnyEvent
DAv2	Depth Anything V2
DAVIS	Dynamic and Active-pixel Vision Sensor
DINO	Self-Distillation with No Labels
DNN	Deep Neural Network
DPT	Dense Prediction Transformer
DSEC	A Stereo Event Camera Dataset for Driving Scenarios
DVS	Dynamic Vision Sensor
E2VID	Events-to-Video Reconstruction Network
ESim	Event Camera Simulator
ET-Net	Event-based Video Reconstruction Transformer Network
EventScape	Synthetic Event-Camera Driving Dataset
FCN	Fully Convolutional Network
FLIR	Forward-Looking Infrared
FullyConv	Fully Convolutional
GPU	Graphics Processing Unit
GT	Ground Truth
HDR	High Dynamic Range
LiDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory
MDE	Monocular Depth Estimation
MiDaS	Monocular Depth Estimation Model
MLP	Multi-Layer Perceptron
MVSEC	Multi Vehicle Stereo Event Camera Dataset
MVS	Multi-View Stereo
RGB	Red, Green, Blue
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RMSE log	Logarithmic Root Mean Squared Error
SAM	Segment Anything Model
SDE	Stereo Depth Estimation
SI log	Scale-Invariant Logarithmic Error
Sq Rel	Squared Relative Error
VFM	Vision Foundation Model
ViT	Vision Transformer
ViT-B	Vision Transformer Base
ViT-L	Vision Transformer Large
ViT-S	Vision Transformer Small

Abbreviation	Definition
VLP-16	Velodyne VLP-16 LiDAR Sensor
VR	Virtual Reality
VRAM	Video Random Access Memory

1

Introduction

Computer vision is a field that allows computers to perceive and understand the world. It is widely used in applications such as autonomous driving, robotics, parking assistance, facial recognition, and virtual and augmented reality [38]. Most computer vision methods rely on frame-based cameras, like conventional RGB sensors, which capture images at fixed intervals and process them for tasks such as object detection and recognition.

Event cameras are a recent type of sensor that work asynchronously. Instead of capturing full images at fixed intervals, they record changes in brightness at each pixel as they happen. This gives them useful properties such as high temporal resolution, low latency, high dynamic range, and low power consumption [9]. These features make them particularly interesting for tasks that require fast or robust perception.

Depth estimation, the main focus of this thesis, aims to predict the distance of each pixel from the camera. Combining event cameras with depth estimation is especially promising: their high temporal resolution helps track fast motion, and their high dynamic range improves performance in challenging lighting.

At the same time, event-based depth estimation is challenging. Event data is sparse, asynchronous, and fundamentally different from RGB images, making it hard to process directly. Large annotated datasets are limited [2], and the sparse nature of events makes depth reconstruction more difficult than with dense frame-based images. Moreover, Vision Foundation Models (VFMs) trained on RGB images expect dense, synchronous, and spatially coherent inputs. Feeding raw or standard event representations into these models leads to poor results.

This motivates learning an event representation instead of relying only on fixed representations. Event data is sparse, asynchronous, and only captures changes in brightness, while RGB-based VFMs are trained on dense images with spatially coherent intensity patterns. A learned representation can adapt the event data into a format that better matches the input space expected by the frozen VFM. By doing this, we can reuse the generalization ability of existing foundation models even when annotated event data is limited.

In this thesis, we investigate whether learning such an event representation allows a frozen RGB-based VFM to perform accurate event-based depth estimation. Our goal is to bridge the gap between event data and the input requirements of existing VFMs, enabling high-quality depth predictions in challenging scenarios.

The thesis is organized as follows. Chapter 1 introduces the research problem and motivation. Chapter 2 provides the necessary background on depth estimation, neural networks, vision transformers, vision foundation models, event cameras, and event representations. Chapter 3 presents the main work as a scientific article, including the proposed method, experiments, and results.

1.1. AI Statement

AI tools were used during the development process primarily to accelerate repetitive tasks, such as generating training configurations and assisting with scripting. All generated code was carefully reviewed and verified before use and was not adopted without validation. For writing, the content and ideas presented are my own. AI was only used to assist with identifying and correcting grammar and spelling issues, without contributing to the content of this thesis.

2

Background

In this chapter, we explain several fundamental concepts that form the foundation of the core work of this thesis. We begin by introducing depth estimation and explaining its purpose and usefulness. Next, we outline the fundamentals of neural networks and deep learning, including their capabilities and key components. We then explore topics more directly related to this thesis, including vision transformers and vision foundation models. Finally, we introduce event cameras and explain how event representations are used.

2.1. Depth Estimation

Depth estimation is the task of predicting a dense, per-pixel depth map from a given image. Specifically, for each pixel, we estimate a depth value that represents the distance of the corresponding scene point from the camera. Figure 2.1 shows an example of monocular depth estimation, where the first row depicts the input RGB image and the second row shows the estimated depth map. We will explain the term monocular in later sections.

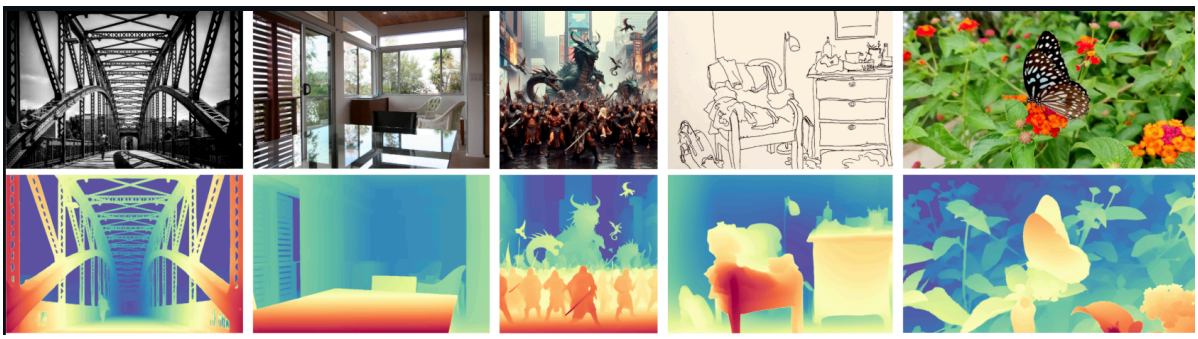


Figure 2.1: Example of monocular depth estimation using Depth Anything V2, adapted from [41]. The top row shows input images from different domains, including natural scenes, synthetic content, sketches, and high dynamic range imagery. The bottom row shows the corresponding predicted depth maps, where warmer colors indicate closer regions and cooler colors indicate farther regions. This illustrates the ability of the model to generalize across diverse visual inputs.

2.1.1. Use Cases

Depth estimation is an important and widely researched topic in computer vision, with many applications in everyday life. A major application is in autonomous driving, where cameras are used to estimate the distance to surrounding objects [33], enabling safe navigation and obstacle detection. Another important application is in robotics, where depth estimation allows robots to perceive and navigate through their environment [4, 43]. Simon et al. [34] introduce a method that enables drones to estimate depth using cameras and subsequently fuse this information into a 3D reconstruction, as shown in Figure 2.2. Additionally, depth estimation is used in virtual and augmented reality [10], where it enables

the estimation of scene geometry for creating immersive virtual environments.

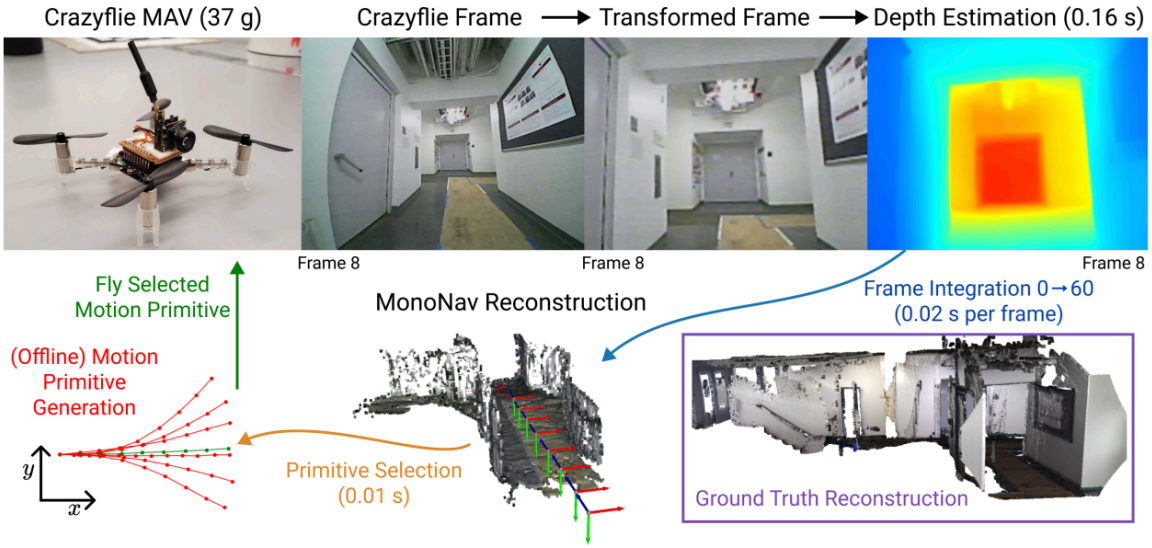


Figure 2.2: Depth estimation applied to drone-based 3D reconstruction, adapted from [34]. The drone first captures monocular image frames, predicts depth for each frame, and then fuses these depth estimates over time to reconstruct the surrounding 3D environment. This example illustrates how depth estimation can support spatial perception and navigation in robotics.

2.1.2. Types of Depth Estimation

There are many ways to estimate the depth of an environment. Early approaches rely on specialized sensors, such as ultrasonic depth estimation, where sound waves are emitted and reflected off surfaces, with the delay encoding the distance. A more accurate approach uses light-based sensors, such as LiDAR, where emitted light is reflected back and the distance is computed based on the time of flight.

However, with the rise of deep learning techniques, cameras have become an important tool for estimating depth. Instead of relying on explicit measurements using light or sound, a neural network can be trained to predict depth directly from visual input. For camera-based depth estimation, which is the focus of this thesis, several approaches exist, including monocular, stereo, multi-view, and event-based methods.

Monocular Depth Estimation

Monocular depth estimation (MDE) refers to estimating depth from a single image. An example is shown in Figure 2.1. An important limitation of MDE is that it is only accurate up to scale. This means that relative depth between objects can be estimated, but the absolute distance (e.g., in meters) cannot be determined. This ambiguity arises because a single image can correspond to infinitely many possible 3D scenes that produce the same projection. This ambiguity is illustrated in Figure 2.3.

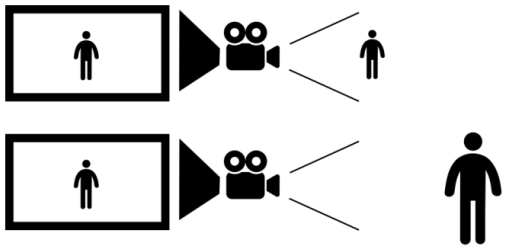


Figure 2.3: Illustration of scale ambiguity in monocular depth estimation, adapted from [15]. A smaller person close to the camera can appear similar in image size to a taller person farther away, making the absolute distance ambiguous from a single image. This shows why monocular depth estimation can recover relative scene structure but cannot directly determine metric scale without additional information.

Stereo Depth Estimation

To overcome the scale ambiguity present in MDE, we can use two images to estimate depth more accurately, a method known as stereo depth estimation (SDE). Using two cameras, we can leverage the difference between the two images of the same scene to calculate the disparity, which refers to the horizontal offset between corresponding points in the left and right images. Typically, the left image is taken as the reference. This significantly reduces the ambiguity, as only one consistent 3D scene can explain both views. An illustration of SDE is shown in Figure 2.4.

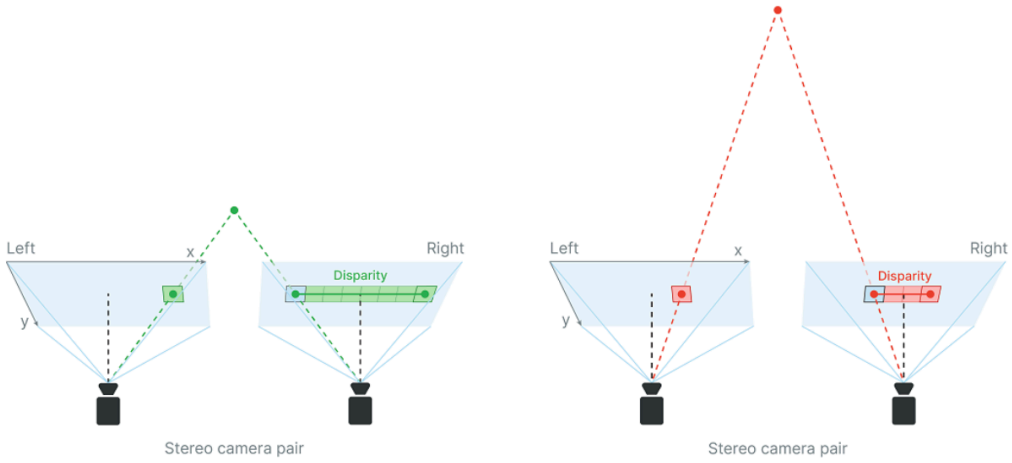


Figure 2.4: Illustration of stereo depth estimation, adapted from [27]. Corresponding points in the left and right images appear at different horizontal positions, producing a disparity that is inversely related to depth. By using two viewpoints of the same scene, stereo depth estimation reduces the scale ambiguity present in monocular depth estimation.

Multi-view Depth Estimation

Stereo depth estimation can be extended to more than two images, referred to as multi-view depth estimation. Multi-view stereo (MVS) methods reconstruct dense 3D geometry from multiple images captured from different viewpoints [42]. By using multiple viewpoints, we can be more confident about the geometry of the 3D scene, as additional views provide more constraints compared to only two images. The downside of this approach is that it requires a more complex setup and increased computational resources, in exchange for improved accuracy and robustness.

2.1.3. Depth Representations

Depth can be represented in different forms, as each representation has its own advantages depending on the scenario. We will discuss a few common depth representations: disparity, metric depth, inverse depth, and relative depth.

Disparity

In computer vision, disparity is commonly used in stereo camera setups. When two cameras are placed next to each other, the horizontal shift between corresponding pixels in the left and right images is called the disparity, as shown in Figure 2.4.

When the baseline T, the focal length f, and the disparity are known, the depth of a 3D point can be computed through triangulation. This relationship is illustrated in Figure 2.5, where the depth Z is inversely proportional to the disparity.

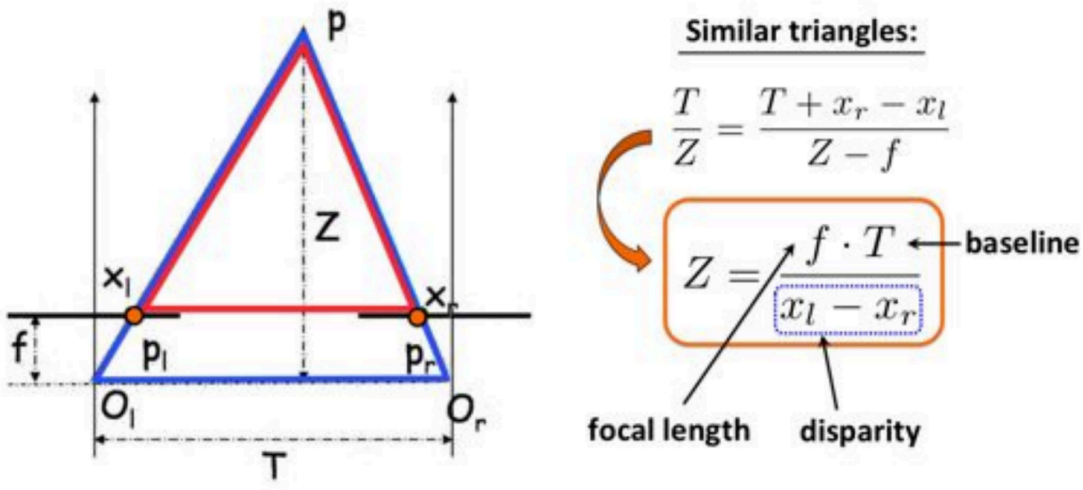


Figure 2.5: Illustration of stereo geometry, adapted from [8]. Given the camera baseline T, focal length f, and disparity $x_l - x_r$, the depth Z of a 3D point can be computed by triangulation. This relationship shows that larger disparities correspond to closer objects, while smaller disparities correspond to farther objects.

Metric Depth

Metric depth, or absolute depth, refers to a representation where each pixel in an image is assigned a value corresponding to its true physical distance from the camera, typically expressed in units such as meters or centimeters. Unlike disparity, where each pixel represents the horizontal shift between corresponding points in stereo images, metric depth directly encodes absolute distances and is therefore used for precise measurements and real-world spatial understanding.

Inverse Depth

Inverse depth is a representation where each pixel value is the inverse of the true depth. This is typically expressed as $d = \frac{1}{z}$, where z is the real depth and d is the inverse depth. This representation has the advantage that it places more emphasis on closer objects and less on farther ones, which is often beneficial in computer vision tasks.

Relative Depth

Relative depth is similar to metric depth, but instead of encoding the absolute distance for each pixel, it represents the relative ordering of depths in the scene. For example, if pixel a has a smaller depth value than pixel b, this indicates that pixel a is closer to the camera than pixel b, without specifying the exact distance.

The advantage of relative depth is that it does not require a fixed scale and can represent depth relationships across a wide range of distances. This makes it useful in scenarios where the absolute scale is unknown or varies significantly, such as when depth ranges from centimeters to kilometers.

The combination of relative and inverse depth is commonly used in modern monocular depth estimation methods. For example, Depth Anything V2 (DAv2) [41] predicts relative inverse depth rather than absolute metric depth. Similarly, MiDaS [30] and DPT [29] also estimate depth in a relative inverse depth space, where the predictions preserve the structure and ordering of the scene but do not directly correspond to metric distances.

2.1.4. Evaluation Metrics for Depth Estimation

To evaluate the predicted depth maps, we use several standard metrics for depth estimation. Let d_i be the predicted depth at pixel i, d_i^* the corresponding ground truth depth, and N the number of valid pixels. We use the following evaluation metrics: absolute relative error (Abs Rel), squared relative error

(Sq Rel), root mean squared error (RMSE), logarithmic RMSE (RMSE log), scale-invariant logarithmic error (SI log), and threshold accuracies $\delta < 1.25$, $\delta < 1.25^2$, and $\delta < 1.25^3$.

The absolute relative error (Abs Rel) measures the relative difference between the prediction and the ground truth:

$$\text{Abs Rel} = \frac{1}{N} \sum_i \frac{|d_i^* - d_i|}{d_i^*}. \tag{2.1}$$

The squared relative error (Sq Rel) is similar, but penalizes larger errors more strongly:

$$\text{Sq Rel} = \frac{1}{N} \sum_i \frac{(d_i^* - d_i)^2}{(d_i^*)^2}. \tag{2.2}$$

The root mean squared error (RMSE) measures the average absolute error in depth space:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_i (d_i^* - d_i)^2}. \tag{2.3}$$

We also report the logarithmic RMSE (RMSE log), which computes the error in log-depth space:

$$\text{RMSE log} = \sqrt{\frac{1}{N} \sum_i (\log d_i^* - \log d_i)^2}. \tag{2.4}$$

In addition, we use the scale-invariant logarithmic error (SI log), which reduces the influence of global scale differences between the prediction and ground truth:

$$\text{SI log} = \frac{1}{N} \sum_i e_i^2 - \frac{1}{N^2} \left(\sum_i e_i \right)^2, \quad e_i = \log d_i^* - \log d_i. \tag{2.5}$$

Finally, we report threshold accuracies. These measure the percentage of pixels for which the prediction is within a certain ratio of the ground truth:

$$\delta_i = \max \left(\frac{d_i^*}{d_i}, \frac{d_i}{d_i^*} \right). \tag{2.6}$$

We report the percentage of pixels satisfying $\delta_i \leq 1.25$, $\delta_i \leq 1.25^2$, and $\delta_i \leq 1.25^3$.

For the error metrics, lower values indicate better performance. For the threshold accuracies, higher values indicate better performance.

2.2. Neural Networks

Neural networks consist of artificial neurons. These artificial neurons are inspired by neurons in the human brain, where they receive input, process it, and then pass the processed signal further on. When multiple artificial neurons are combined and arranged in layers, they form an artificial neural network. Figure 2.6 shows the comparison between a biological neuron and an artificial neuron.

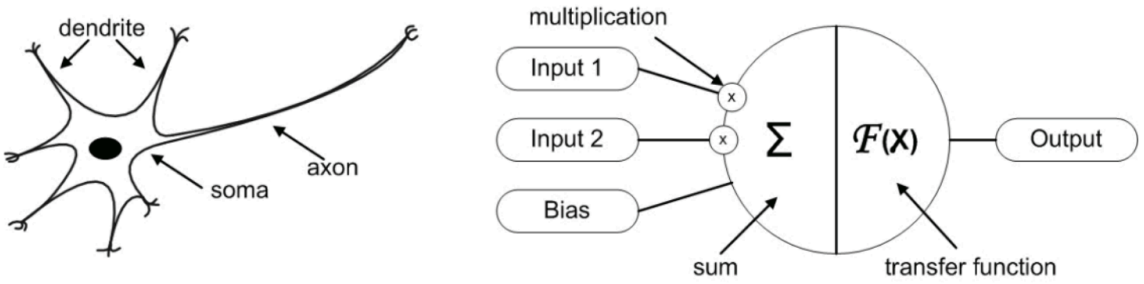


Figure 2.6: Comparison between a biological neuron and an artificial neuron. In the biological neuron (left), dendrites receive input signals, which are processed in the soma and transmitted through the axon. The artificial neuron (right) models this process by computing a weighted sum of input signals, adding a bias term, and applying a nonlinear activation (transfer function) to produce an output [21].

To explain the mathematics behind a neural network in more detail, we give an example in Figure 2.7. Here, a simple neural network is depicted with only a few connections to keep the example clear.

The first layer that processes the input is called the input layer. The last layer that produces the result is called the output layer. All layers in between are referred to as hidden layers.

For example, consider the input layer with three inputs x_1, x_2, x_3. Each connection has an associated weight, such as w_1, w_2, w_3, and each neuron has a bias b_1, b_2, b_3. The inputs are multiplied by their corresponding weights, the bias is added, and the result is passed through a transfer function F_i. This produces the outputs n_1, n_2, n_3, n_4, which are then passed to the next layer.

This process can be written mathematically as:

$$n_1 = F_1(w_1 x_1 + b_1) \tag{2.7}$$
$$n_2 = F_2(w_2 x_2 + b_2) \tag{2.8}$$
$$n_3 = F_2(w_2 x_2 + b_2) \tag{2.9}$$
$$n_4 = F_3(w_3 x_3 + b_3) \tag{2.10}$$

The next hidden layer combines these outputs using new weights q_i and biases:

$$m_1 = F_4(q_1 n_1 + q_2 n_2 + b_4) \tag{2.11}$$
$$m_2 = F_5(q_3 n_3 + q_4 n_4 + b_5) \tag{2.12}$$

Finally, the output layer produces the final result:

$$y = F_6(r_1 m_1 + r_2 m_2 + b_6) \tag{2.13}$$

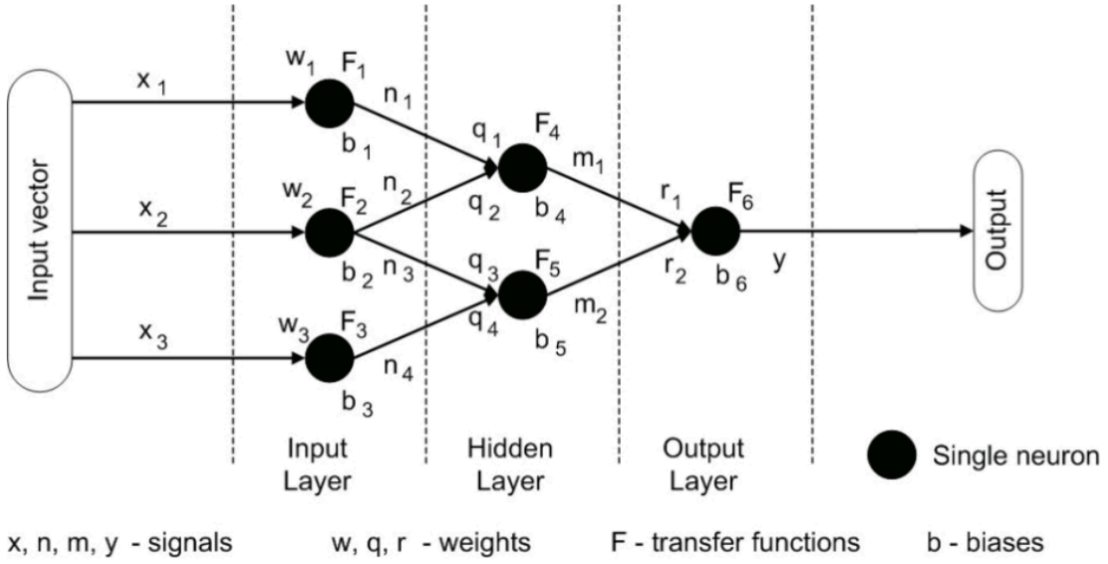


Figure 2.7: Simplified neural network with an input layer, hidden layer, and output layer, adapted from [21]. Input values are multiplied by learnable weights, combined with bias terms, and passed through transfer functions to produce intermediate activations and the final output. This illustrates how neural networks transform input features step by step into a prediction.

Transfer Functions

Transfer functions, also called activation functions, are a key part of neural networks. They introduce non-linearity into the model. Without this non-linearity, a neural network would only be able to represent linear relationships, which is not sufficient for most real-world problems.

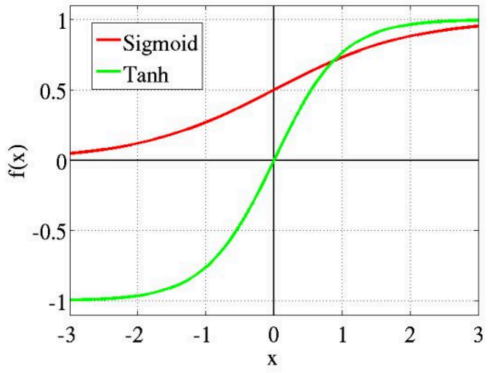
By applying a nonlinear function after each layer, the network can learn more complex mappings between inputs and outputs. Common examples of transfer functions include ReLU, sigmoid, and hyperbolic tangent functions, defined as:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.14}$$

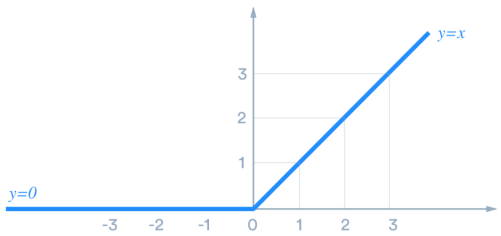
$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.16}$$

These functions are illustrated in Figure 2.8.



(a) Tanh (green) and sigmoid (red) activation functions.



(b) ReLU activation function.

Figure 2.8: Common transfer functions used in neural networks, adapted from [35]. The sigmoid and hyperbolic tangent functions compress inputs into bounded ranges, while ReLU keeps positive values and suppresses negative values. These nonlinear functions allow neural networks to model complex relationships beyond simple linear mappings.

2.2.1. Training

A key technique used for training neural networks is backpropagation [32]. Backpropagation efficiently computes how much each parameter contributes to the loss. During training, a loss function is defined to measure the difference between the prediction and the ground truth. The goal is to minimize this loss by computing gradients and updating the network parameters in the opposite direction of the gradient, moving the model toward a lower loss.

To give an example, suppose the inputs x_1, x_2, x_3 from Figure 2.7 represent the temperature, vibration level, and load of a mechanical component. The output y from Equation 2.13 is a prediction between 1 and 0, indicating whether the component is likely to fail.

To train the network, we need training data. As an example, consider a sample with values $80°C$, high vibration, and high load, with the label *failure*, or 1. This label is called the ground truth. During training, we input these values into the network, and it might output something like 0.3. This is far from the correct result, since the ideal output should be close to 1.

To train the network, we pass the inputs through the neural network and compute a loss based on the output. For example, using an L_2 loss:

$$\text{loss} = (1 - 0.3)^2 \tag{2.17}$$

The L_2 loss measures the squared difference between the predicted value and the ground truth. In this case, the prediction is 0.3 while the correct value is 1, resulting in a relatively large loss. The squaring ensures that larger errors are penalized more strongly and that the loss is always positive. In general, the L_2 loss can be written as:

$$L = (y_{\text{true}} - y_{\text{pred}})^2 \tag{2.18}$$

Next, the derivative of the loss is computed. For the L_2 loss, the derivative with respect to the prediction is:

$$\frac{\partial L}{\partial y_{\text{pred}}} = 2(y_{\text{pred}} - y_{\text{true}}) \tag{2.19}$$

This derivative indicates how sensitive the loss is to changes in the network output. Using backpropagation, this information is propagated through the network to compute how each weight contributes to the loss. The weights (e.g., w_1, w_2, w_3) are then updated in a direction that reduces the loss.

By repeating this process many times, the network gradually learns to produce outputs closer to 1 for similar inputs. Figure 2.9 illustrates this process, where the loss is iteratively reduced until it converges.

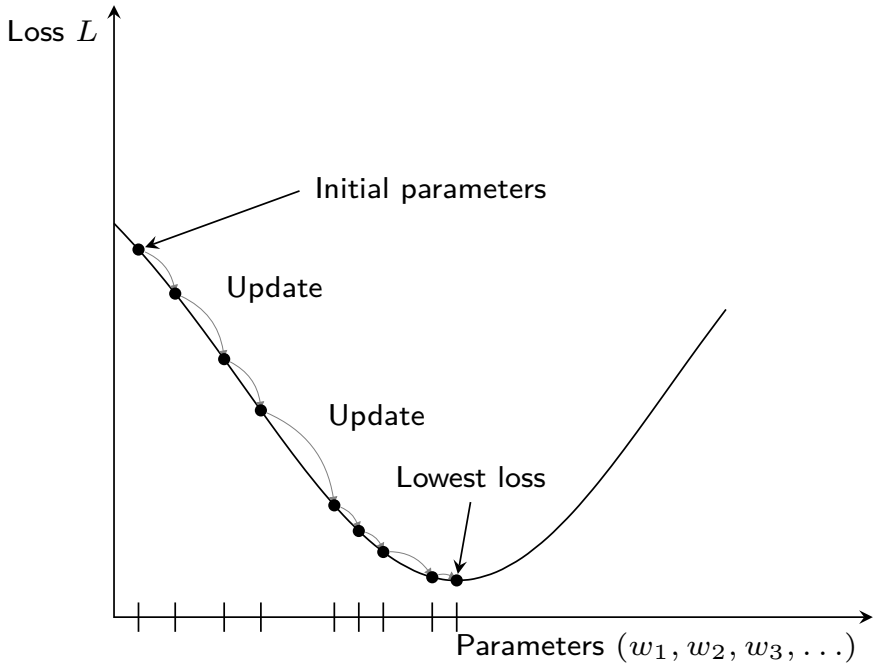


Figure 2.9: Illustration of neural network training by iterative optimization, adapted from [36]. Starting from an initial set of weight parameters, the network computes gradients using backpropagation and updates the weights step by step to reduce the loss. This process gradually moves the model toward weight parameter values that produce more accurate predictions.

2.2.2. Convolutional Neural Networks

A neural network layer typically operates on a vector of numerical data, as shown in the example in Figure 2.7. When applying this directly to images, the image would first need to be converted into a vector. However, this quickly becomes impractical, as the number of inputs grows very large. For example, a 400×400 RGB image results in $400 \times 400 \times 3$ input values.

To effectively apply deep learning to images, convolutional neural networks (CNNs) are used [25]. Instead of treating the image as a long vector, CNNs take advantage of the spatial structure of the image.

A convolutional neural network works by sweeping a convolutional kernel over the image. This kernel is a small patch of trainable weights that slides across the image and computes a weighted sum at each location. In this way, the network extracts local features such as edges, textures, or patterns. The application of a convolution kernel is shown in Figure 2.10.

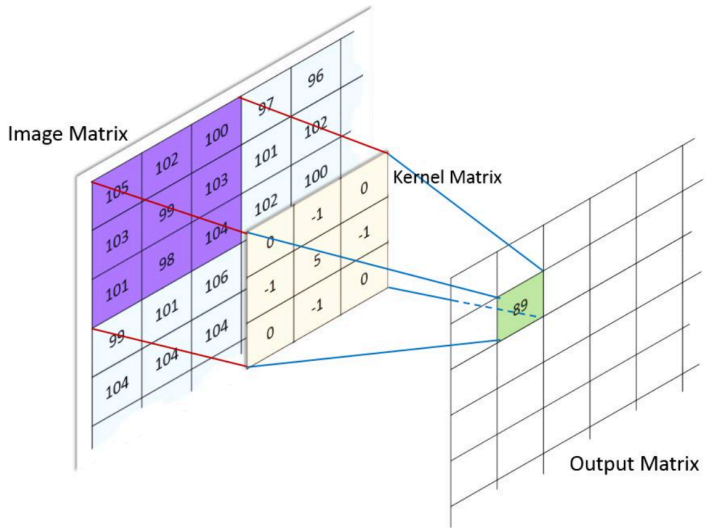


Figure 2.10: Illustration of the convolution operation. A convolution kernel slides over the input image and computes a weighted sum of the kernel values (Kernel Matrix) and the local image patch (Image Matrix). The resulting value is stored at the corresponding position in the output feature map (Output Matrix). In this example, the highlighted image region is multiplied element-wise with the kernel matrix to produce the output value 89. Image adapted from [40].

By applying multiple kernels and stacking several layers, the network can learn increasingly complex features. Early layers typically detect simple structures such as edges and textures, while deeper layers combine these into higher-level semantic representations such as object parts and complete objects [22].

In RGB-based monocular depth estimation, convolutional neural networks have been widely used to predict depth from a single RGB image. Early work by Eigen et al. [7] showed that CNNs can learn useful depth cues from images. Laina et al. [24] further improved supervised depth estimation using a deeper fully convolutional residual network for dense per-pixel prediction. In addition to supervised methods, self-supervised approaches such as Monodepth2 [14] reduce the need for ground truth depth by training with image reconstruction losses from stereo image pairs or monocular video. Although these methods differ in their training strategy, they show that CNN-based models can learn important visual cues for depth estimation, such as perspective, object layout, and local image structure.

Dilated Convolution

There exist many variants of the standard convolution operation. One commonly used variant, which is also relevant to this thesis, is dilated convolution. An illustration is shown in Figure 2.11.

Dilated convolution increases the receptive field of the convolution kernel without increasing the number of trainable parameters. The receptive field refers to the area of the input image that a neuron or kernel can observe. This is achieved by inserting gaps between the kernel elements, allowing the convolution to capture information from a larger region of the image while keeping the same kernel size.

Compared to standard convolution, dilated convolution is particularly useful in tasks where larger context is important, such as semantic segmentation and depth estimation.

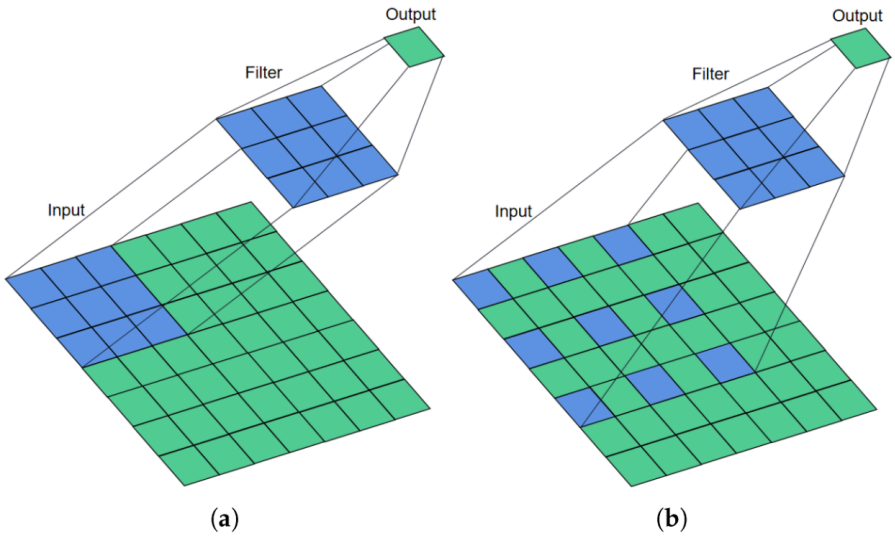


Figure 2.11: Illustration of standard and dilated convolution. In standard convolution (a), the kernel operates on neighboring pixels directly, resulting in a smaller receptive field. In dilated convolution (b), gaps are introduced between kernel elements, allowing the kernel to cover a larger spatial region without increasing the kernel size or the number of parameters. Image adapted from [16].

2.3. Transformers

Transformer architectures have become a fundamental component in modern deep learning. They were originally introduced by Vaswani et al. [37] for neural machine translation. While transformers were initially designed for machine translation, they are now widely used across many domains. In particular, they form the foundation of large language models, where they are used to model long-range dependencies in text. The architecture of the Transformer is shown in Figure 2.12.

Self-attention

The core idea behind transformers is the use of the self-attention mechanism within the encoder and decoder blocks. The goal of self-attention is to allow each element in a sequence, e.g., a word in a sentence, to attend to all other elements in that same sequence. This enables the model to capture global context, rather than relying only on local or sequential information.

As a result, transformers can process all elements in a sequence in parallel. This differs from earlier sequence models, such as LSTMs [18], which process the sequence step by step and therefore depend on the previous hidden state. Parallel processing makes transformers more efficient to train and allows them to better capture long-range dependencies, especially in long sequences.

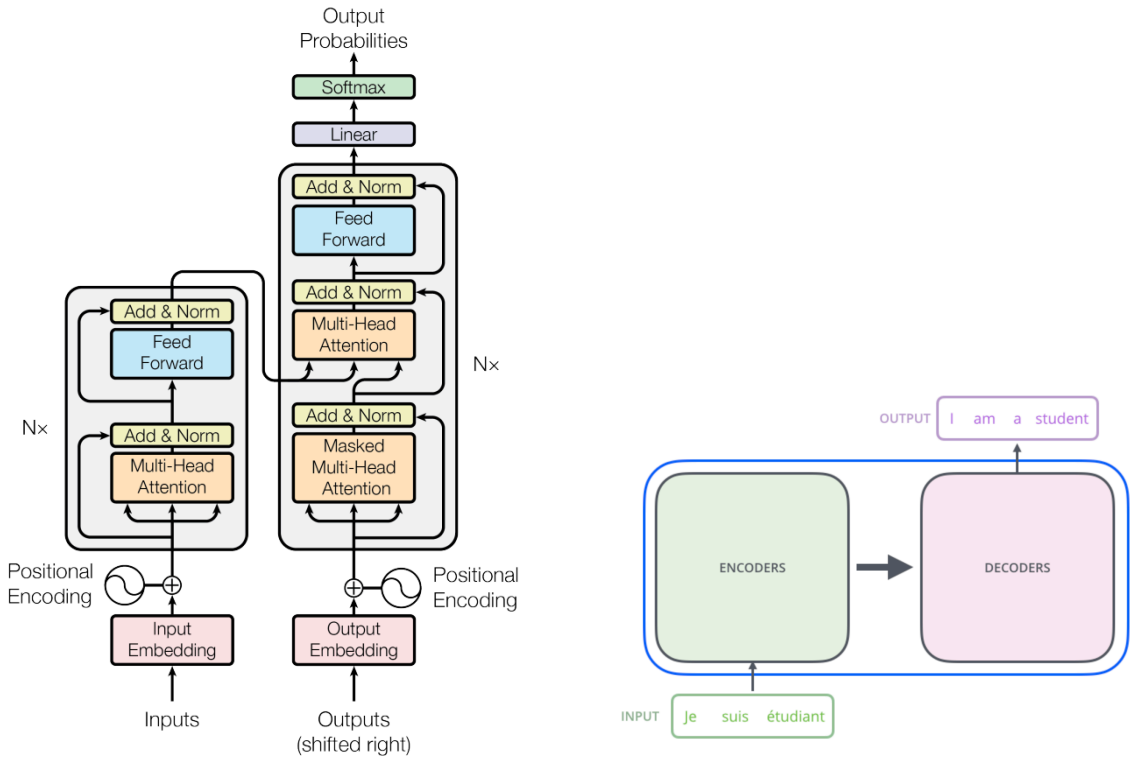
The self-attention mechanism works by projecting each input token into three different vectors: a query, a key, and a value. First, each input token is embedded into a vector representation. This embedding is then multiplied by three learnable weight matrices, W^Q, W^K, and W^V, to obtain the corresponding query, key, and value vectors. This process is illustrated in Figure 2.13.

After the query, key, and value vectors are computed for all input tokens, the query and key vectors are compared to compute attention scores. These scores indicate how relevant each token is to the other tokens in the sequence. The scores are then converted into weights using a softmax function.

The value vectors are multiplied by these weights and combined into a new representation. As a result, each output vector contains information from its own token, but also from other tokens that the model considers relevant.

The scaled dot-product attention is defined as:

$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{2.20}$$



(a) Original Transformer architecture from Vaswani et al. [37], consisting of N_x encoder blocks and N_x decoder blocks.

(b) Simplified Transformer architecture adapted from [17], showing the repeated encoder and decoder blocks for translating a French sentence to English.

Figure 2.12: Comparison between the original Transformer architecture and a simplified encoder-decoder representation, adapted from [37]. The original model consists of repeated encoder and decoder blocks, while the simplified diagram shows how an input sequence is encoded and then decoded into an output sequence. This illustrates how transformers process sequences using stacked attention-based blocks.

where Q, K, and V are the query, key, and value matrices, and d_k is the dimensionality of the key vectors. The softmax converts the attention scores into weights. These weights are then applied to the value vectors to obtain the output representation Z. This process is shown in Figure 2.14a.

The resulting vectors are passed to the feed-forward network and then to the next encoder layer. This process is repeated for each encoder block, allowing the model to gradually build richer contextual representations. This is shown in Figure 2.14b.

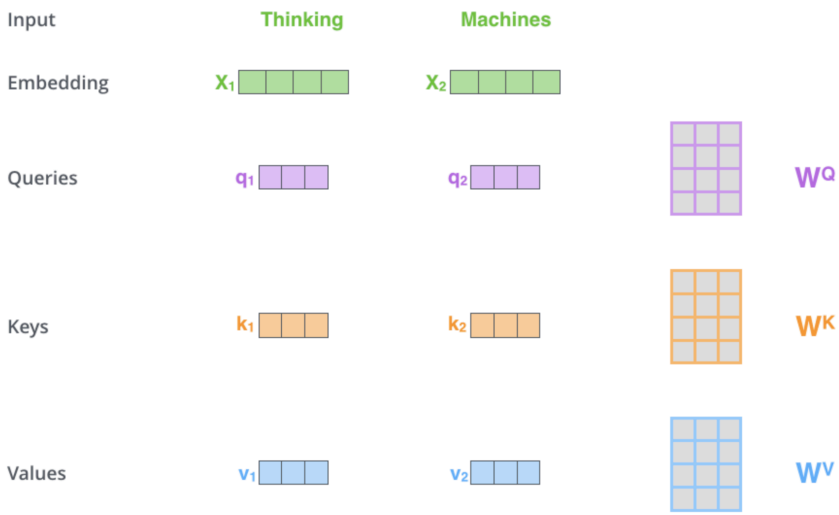
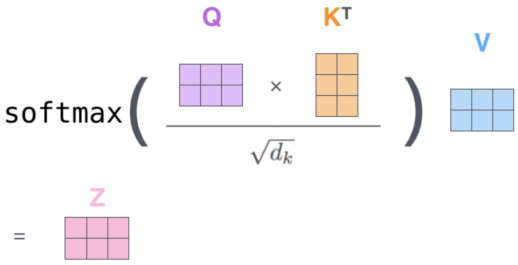
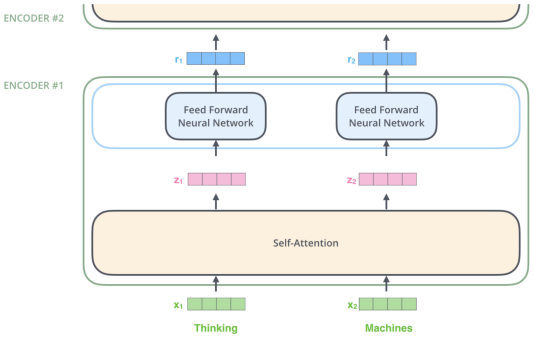


Figure 2.13: Illustration of the query, key, and value projections in self-attention, adapted from [1]. Each input token embedding is multiplied by three learnable weight matrices, W^Q, W^K, and W^V, producing a query, key, and value vector for each token. These projections allow the model to compute how strongly each token should attend to the others.



(a) Scaled dot-product attention. The attention output Z is computed by comparing queries Q with keys K, normalizing the scores with softmax, and applying the result to the values V.



(b) Output of one Transformer encoder layer. The self-attention output is passed through a feed-forward network, producing representations that are passed to the next encoder layer.

Figure 2.14: Overview of how self-attention produces intermediate token representations in a Transformer encoder, adapted from [1]. The scaled dot-product attention compares queries with keys, normalizes the scores with softmax, and applies the resulting weights to the values. The output is then passed through a feed-forward network, allowing the encoder to build increasingly contextual representations across layers.

2.3.1. Vision Transformers

Transformers are not limited to sequence-based inputs such as text. They have also been successfully applied to image data. This application was introduced by Dosovitskiy et al. [5], where transformers were adapted for image recognition tasks.

Since then, Vision Transformers (ViTs) have become a standard approach for achieving high accuracy in computer vision. These models apply the transformer architecture to images, enabling them to capture global context in a similar way as in sequence modeling. The architecture of ViT is shown in Figure 2.15.

In a Vision Transformer, the image is first divided into smaller fixed-size patches. Each patch can be seen as a small image region. These patches are then flattened and projected into vectors, also called patch embeddings. Since the transformer processes the patches as a sequence, positional embeddings are added to preserve information about where each patch came from in the original image. The resulting sequence of patch embeddings is passed through a Transformer encoder, where self-attention allows each patch to exchange information with all other patches. As a result, the model

can capture both local information from individual patches and global relationships across the full image.

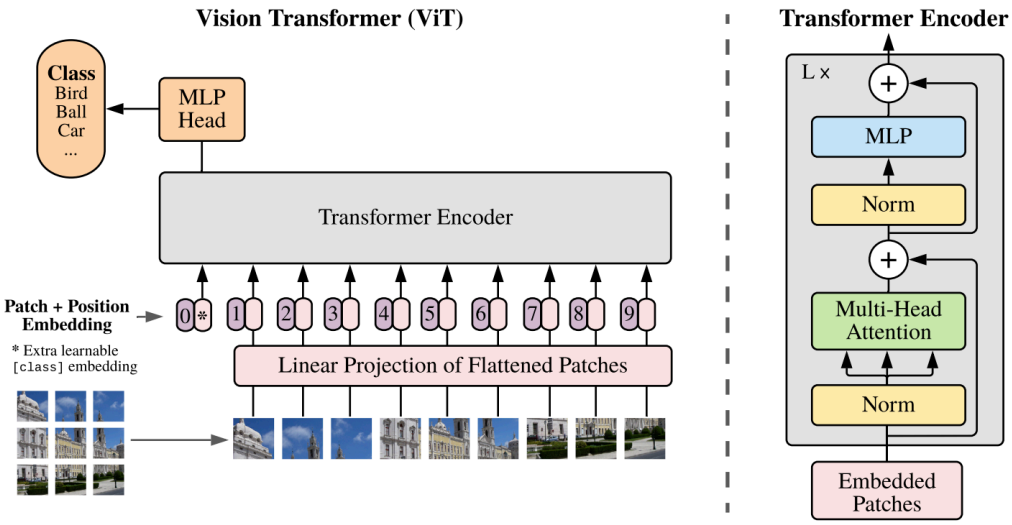


Figure 2.15: Overview of the Vision Transformer architecture, adapted from [5]. The input image is divided into patches, projected into patch embeddings, combined with positional embeddings, and processed by a Transformer encoder. This allows the model to treat image patches as a sequence and capture global relationships across the image.

2.4. Vision Foundation Models

Vision foundation models are general-purpose neural networks trained on large-scale datasets. Their goal is to learn rich and transferable representations that can be applied to a wide range of downstream tasks.

Most modern vision foundation models are based on Vision Transformers, due to their strong performance and ability to capture global context. Examples include DINO [3], a self-supervised representation learning method; SAM [20], which is designed for image segmentation; and Depth Anything V2 [41], which focuses on depth estimation.

2.4.1. Depth Anything V2

Depth Anything V2 is a vision foundation model for monocular depth estimation. Its goal is to predict a dense depth map from a single RGB image while generalizing well to unseen scenes and visual domains. In contrast to task-specific depth networks trained on a limited set of real-world depth datasets, Depth Anything V2 can be applied directly to unseen images or datasets without additional task-specific fine-tuning [41].

An important aspect of Depth Anything V2 is its use of synthetic data for supervised training. Real-world depth labels are often noisy because they are obtained from imperfect sensors or reconstruction pipelines. For example, depth sensors can fail on transparent or reflective surfaces and stereo matching can be unreliable in textureless or repetitive regions. These errors are problematic for dense prediction because the model may learn the noise present in the supervision. Synthetic datasets, on the other hand, can provide complete and accurate depth annotations, including for challenging cases such as transparent and reflective surfaces.

However, training only on synthetic data is not sufficient, because synthetic images look different from real images and contain less scene diversity. Depth Anything V2 addresses this problem with a teacher-student training strategy. First, a large teacher model based on a DINOv2-Giant encoder is trained on 595K accurately labeled synthetic images from five datasets. The teacher is then used to predict pseudo-depth labels for 62M unlabeled real images collected from eight large-scale public datasets. Finally, smaller student models with different encoder sizes are trained on these pseudo-labeled real

images. This allows the student models to learn from the teacher's depth predictions while also benefiting from the visual diversity of real-world images [41].

The model architecture follows the Dense Prediction Transformer (DPT) design [29], as shown in Figure 2.16. A Vision Transformer encoder extracts image features with global context, while a convolutional decoder converts the transformer tokens into a dense per-pixel depth prediction. Depth Anything V2 uses DINOv2 [28] encoders at different scales, including small, base, large, and giant variants. This makes it possible to choose between faster lightweight models and larger, more accurate models depending on the application. The model predicts relative inverse depth by default. For applications requiring metric depth, Depth Anything V2 can be further fine-tuned using metric depth labels [41].

In this thesis, Depth Anything V2 with the small backbone is used as the RGB-based vision foundation model for monocular depth estimation. Its strong generalization ability makes it a suitable backbone for studying whether knowledge learned from large-scale RGB data can be reused for event-based depth estimation.

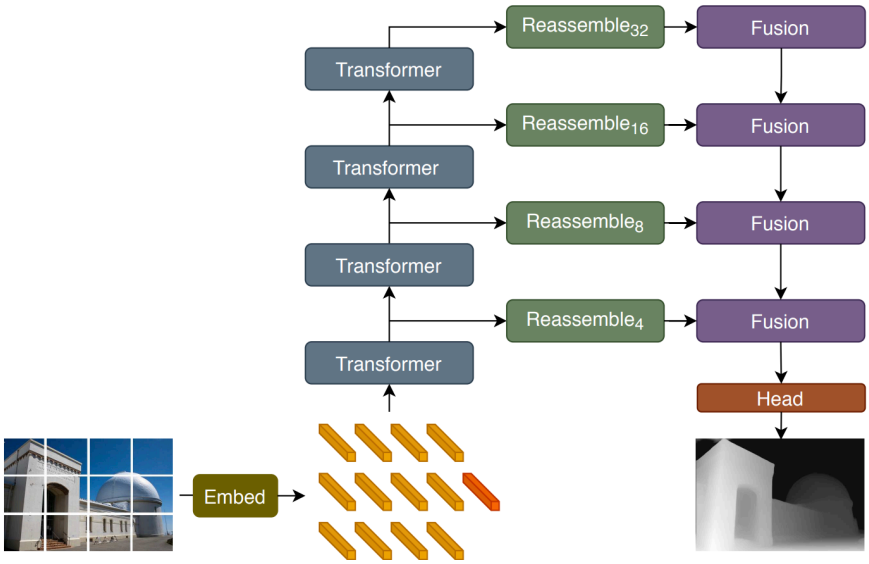


Figure 2.16: Architecture of the Dense Prediction Transformer (DPT) used for dense prediction tasks such as monocular depth estimation, adapted from [29]. The input image is first converted into patch tokens and processed by a Vision Transformer encoder. Tokens from multiple transformer stages are then reassembled into image-like feature maps at different resolutions. These feature maps are progressively combined by fusion blocks and passed to a task-specific prediction head to produce a dense depth map.

2.4.2. Depth AnyEvent

Depth AnyEvent [2] is a recent method for event-based monocular depth estimation. It addresses the limited availability of dense depth annotations for event data by using knowledge from a frame-based Vision Foundation Model (VFM), such as Depth Anything V2.

The main idea is cross-modal distillation. During training, an RGB frame is passed through a frame-based teacher model to predict a dense depth map. This prediction is then used as a proxy label for an event-based student model, which receives the aligned event representation as input and learns to predict depth from events. In this way, knowledge from the RGB domain is transferred to the event domain without requiring dense depth annotations from sensors such as LiDAR. This idea is shown in Figure 2.17.

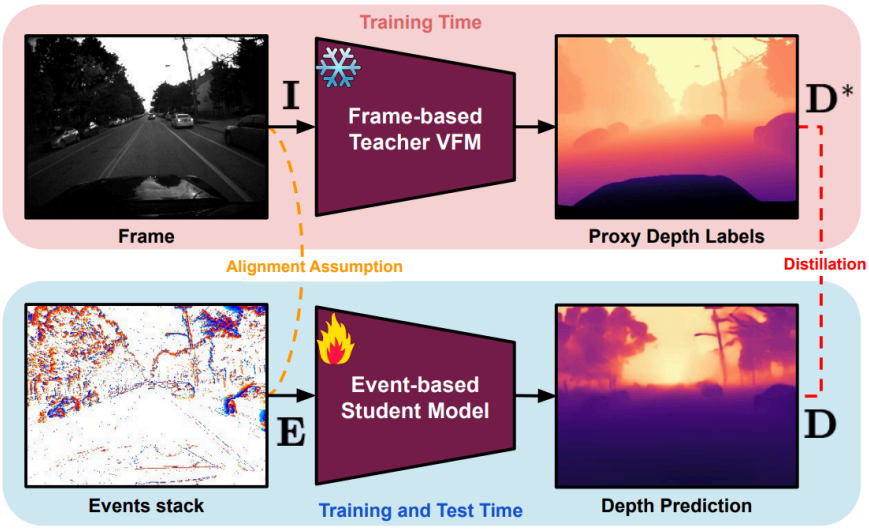


Figure 2.17: Distillation strategy of Depth AnyEvent, adapted from [2]. During training, an RGB frame is processed by a frame-based VFM teacher to generate a dense proxy depth label. An event-based student model receives the aligned event representation and learns to predict a depth map using this proxy supervision. This transfers depth knowledge from RGB-based models to event-based depth estimation.

Depth AnyEvent also adapts Depth Anything V2 directly to event data. Since raw events cannot be processed like RGB images, the events are first converted into the Tencode representation [19], where event polarity and timing are encoded into three channels. The authors also introduce a recurrent variant, DepthAnyEvent-R, which uses information from previous event stacks to better exploit the temporal nature of event streams.

Depth AnyEvent is closely related to this thesis because it also investigates how RGB-based depth foundation models can be used for event-based monocular depth estimation. However, Depth AnyEvent adapts the VFM using a fixed event representation such as Tencode. In contrast, this thesis keeps the Depth Anything V2 backbone frozen and learns the event representation itself, allowing us to study whether a learned representation can better adapt event data to the input space expected by the VFM.

2.5. Event Cameras

Event cameras, or neuromorphic cameras, are a novel way of sensing the real world compared to traditional RGB cameras. One of the first practical event cameras, the Dynamic Vision Sensor (DVS), was introduced by Lichtsteiner et al. [26] in 2008, although the concept of neuromorphic vision sensors dates back even earlier. In recent years, they have gained increasing attention in the computer vision community, especially in combination with deep learning methods.

Neuromorphic sensors operate in a fundamentally different way than standard RGB sensors. Most importantly, event cameras are asynchronous, whereas RGB cameras are synchronous. In a conventional RGB camera, a full image is captured by reading out the entire pixel array at fixed time intervals, producing frame-based images. In contrast, pixels in an event camera operate independently and only generate an event when a change in intensity is detected.

An event is typically represented as a 4-tuple:

$$(p, t, x, y)$$

where p denotes the polarity, indicating whether the intensity increased (positive) or decreased (negative), t is the timestamp of the event, and (x, y) are the pixel coordinates.

Because event camera pixels operate independently and asynchronously, they achieve very high temporal resolution and low latency, typically in the order of microseconds [9]. In addition, they offer a high dynamic range and low power consumption compared to traditional RGB cameras.

These properties make event cameras suitable for applications such as high-speed object detection [11], depth estimation [17], and high dynamic range imaging [39].

2.5.1. Event Representations

Because events are sparse and asynchronous, there are fewer established methods to process them directly. To leverage existing methods designed for image-like inputs, various event representations have been proposed to convert sparse and asynchronous event data into dense, synchronous, image-like formats. These are commonly referred to as event representations.

There are many such representations, including voxel grids, time surfaces, and event frames. In addition, more specialized representations exist. In this thesis, the Tencode event representation [19] plays an important role.

Voxel Grids

Voxel grids, also referred to as event volumes, are one of the most natural dense representations for event data. A voxel grid is a 3D volume where events are accumulated along three axes: time, and the spatial coordinates x and y. Each voxel may have different values, such as counts or the polarity of the event.

The advantage of voxel grids is that they preserve temporal information, unlike 2D representations that collapse time. However, a disadvantage is that they can be memory-intensive and computationally expensive, especially when using a high temporal resolution. An example of a voxel grid is shown in Figure 2.18.

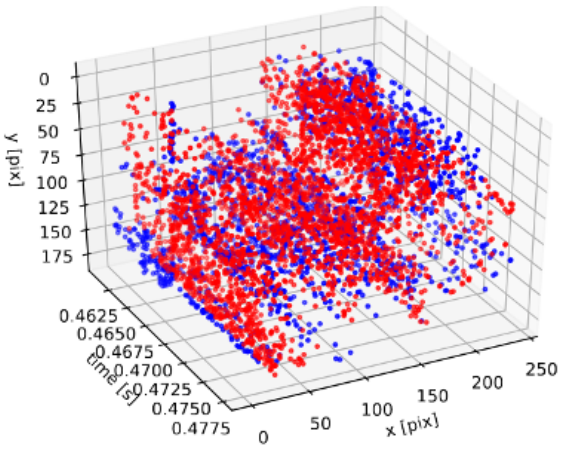


Figure 2.18: Example of a voxel grid representation for event data, adapted from [9]. Events are accumulated over spatial coordinates and time, with red indicating negative intensity changes and blue indicating positive intensity changes. This representation preserves temporal structure better than a single 2D event frame, but can require more memory due to its additional time dimension.

Time Surfaces

A time surface is a 2D representation where each pixel stores a single time value, typically the timestamp of the most recent event at that location [23]. As a result, time surfaces can be seen as images in which the intensity of each pixel depends on the recency of events.

The main advantage of time surfaces is that pixel values can be updated asynchronously while preserving precise temporal information. However, a limitation is that only the timestamp of the most recent event is stored, meaning that earlier events at the same pixel are discarded.

An example of a time surface is shown in Figure 2.19.

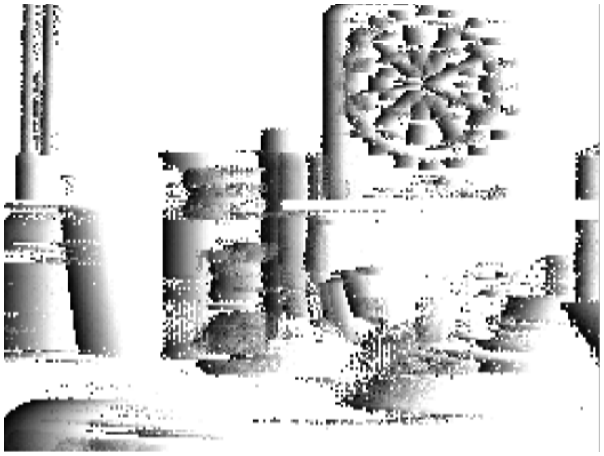


Figure 2.19: Example of a time surface representation for event data, adapted from [9]. Each pixel stores the timestamp of the most recent event at that location, with darker pixels indicating more recent events. This representation preserves event recency in an image-like format, but discards older events at the same pixel.

Event Frames

Event frames, also referred to as 2D histograms, are a 2D image-like event representation where pixel values are obtained by accumulating events over a time window, for example by counting the number of events or summing their polarities.

The main advantage of event frames is that they produce a dense 2D representation, making them directly compatible with traditional CNNs. However, a limitation is that temporal information is lost, as the exact timing of events is not preserved.

An example of an event frame is shown in Figure 2.20.

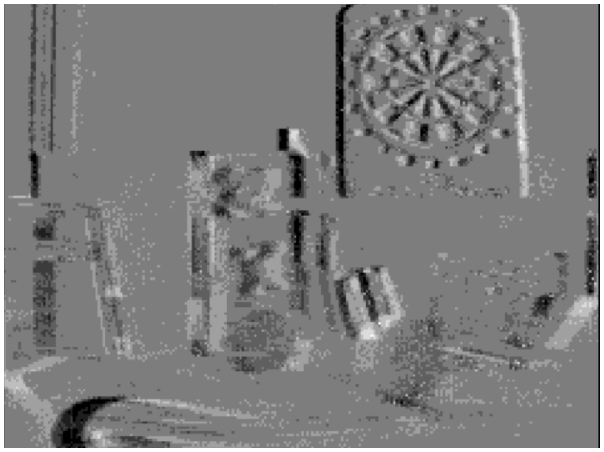


Figure 2.20: Example of an event frame representation, adapted from [9]. Events are accumulated over a fixed time window into a 2D image, where gray indicates no events, black indicates negative polarity, and white indicates positive polarity. This representation is directly compatible with conventional image-based neural networks, but loses precise temporal information.

Tencode

The Tencode event representation was introduced by Huang et al. [19] as a method to convert event-based data into an image-like representation. Given a frame F with three color channels, Tencode maps events to pixel values based on their polarity and timestamp:

$$F(x,y) = \left(255, \frac{255 \cdot (t_{\max} - t)}{\Delta t}, 0\right), \quad \text{for } (x,y,t,+1) \tag{2.21}$$

$$F(x,y) = \left(0, \frac{255 \cdot (t_{\max} - t)}{\Delta t}, 255\right), \quad \text{for } (x,y,t,-1) \tag{2.22}$$

where (x, y, t, p) represents an event with spatial coordinates (x, y), timestamp t, and polarity $p \in \{+1, -1\}$. The variable $t_{\max}$ denotes the timestamp of the most recent event within the time window $[t_{\max} - \Delta t, t_{\max}]$, and Δt is the duration of the time window.

In this representation, positive events are encoded in the red channel and negative events in the blue channel, while the green channel represents the relative timing of the event within the time window. More recent events have higher intensity values, allowing the representation to capture temporal information in a dense image format. An example of Tencode is shown in Figure 2.21.

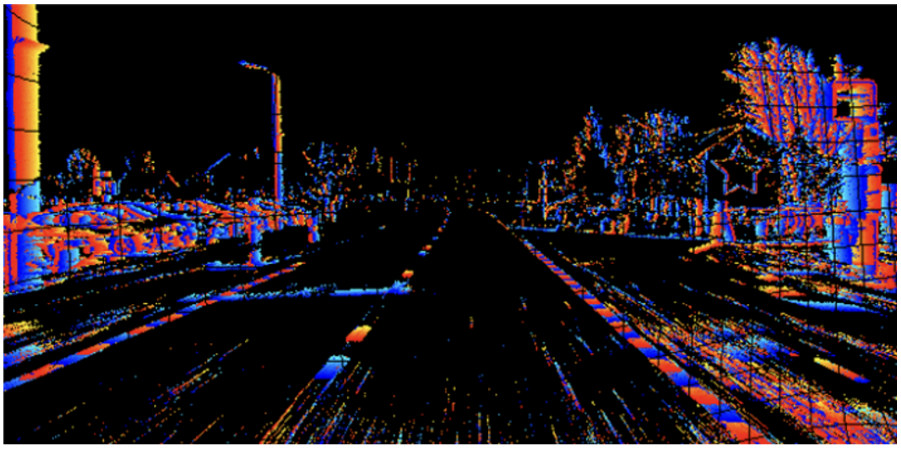


Figure 2.21: Example of the Tencode event representation, adapted from [19]. Positive events are encoded in the red channel and negative events in the blue channel, while the green channel represents the relative timing of events within the time window. Pixels with no events are set to zero. This results in an RGB-like representation that can be directly used as input to models originally designed for image data.

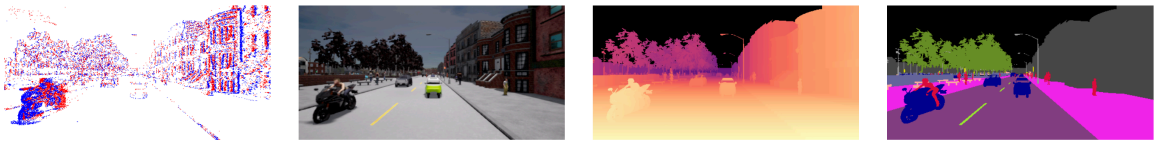
2.6. Datasets

An important requirement for training deep neural networks is the availability of datasets. In the event domain, datasets can generally be divided into two categories: synthetic datasets and real-world datasets. In this section, we first explain what synthetic datasets are and why they are used. We then introduce the real-world datasets used in this thesis: MVSEC [44] and DSEC [13].

2.6.1. Synthetic Datasets

Synthetic datasets are datasets that are not captured directly from the real world. Instead, the data is generated using simulation environments. In the event domain, this means that event data is artificially generated using simulators such as CARLA [6] or ESim [31]. An example of such a dataset is EventScape [12], which is generated using the CARLA simulator. An example is shown in Figure 2.22.

Synthetic datasets are useful because they make it possible to generate large amounts of data with accurate ground truth. For example, by simulating driving scenarios, we can create artificial event streams together with corresponding depth maps. This makes synthetic datasets especially useful for pretraining neural networks before applying them to real-world event data.



(a) events (b) images (c) depth maps (d) segmentation labels

Figure 2.22: Example from the EventScape synthetic dataset, adapted from [12]. From left to right, the figure shows generated event data, an RGB image, a depth map, and semantic segmentation labels for the same simulated driving scene. This illustrates how synthetic datasets can provide multiple aligned modalities with accurate ground truth for training or pretraining.

2.6.2. DSEC

The first dataset used in this thesis is A Stereo Event Camera Dataset for Driving Scenarios (DSEC) [13]. DSEC is a driving dataset recorded in Zurich and contains both daytime and nighttime sequences. The dataset provides event data, RGB images, and ground truth depth obtained from LiDAR scans.

In total, the dataset consists of approximately 26k samples, which are divided into 19k training samples and 7k testing samples. Both the training and testing splits contain a mix of daytime and nighttime sequences.

The event data is captured using two stereo Prophesee Gen3.1 event cameras with a resolution of 640×480. The RGB images are captured using stereo FLIR Blackfly cameras with a resolution of 1440×1080. The ground truth is obtained using a 16-channel Velodyne VLP-16 LiDAR.

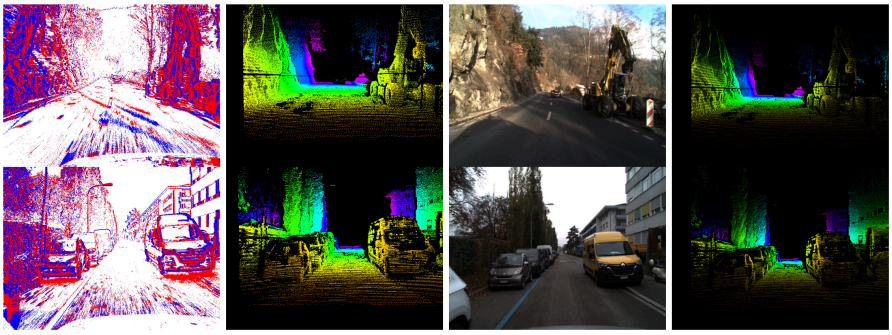


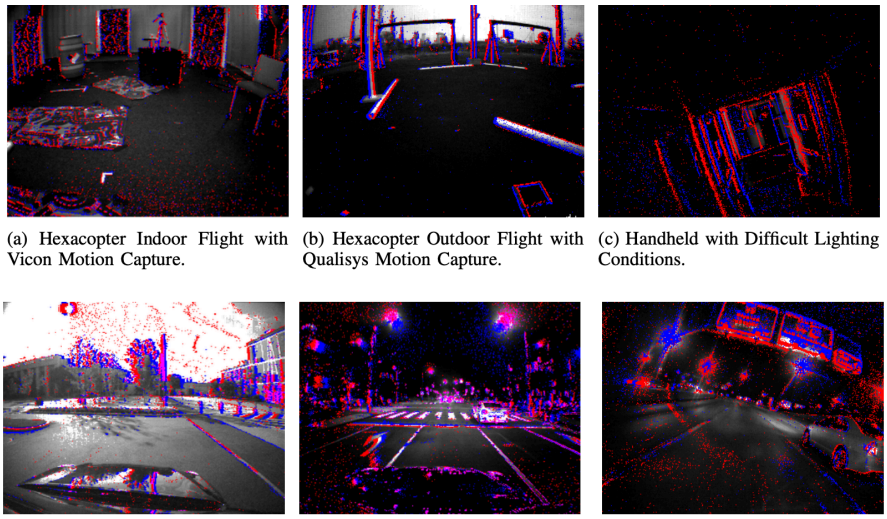
Figure 2.23: Example samples from the DSEC dataset, adapted from [13]. From left to right, the figure shows event data, depth maps aligned with the event camera, RGB images, and depth maps aligned with the RGB camera for different driving scenes. This dataset provides real-world driving data with event streams, RGB images, and LiDAR-based depth ground truth.

2.6.3. MVSEC

The second dataset used in this thesis is the Multi Vehicle Stereo Event Camera Dataset (MVSEC) [44]. Compared to DSEC, MVSEC is an older dataset with a lower spatial resolution. It contains data from different vehicles, including hexacopters, motorcycles, and cars. In this thesis, we only use the car driving sequences.

In total, MVSEC consists of approximately 32k samples, which we split into 12k training samples and 20k testing samples. To keep the comparison consistent with related work, the training set consists of a single daytime sequence, while the testing set contains one daytime sequence and three nighttime sequences.

The event data is captured using stereo DAVIS 346B event cameras with a resolution of 346×260. An important property of this camera is that it includes an integrated RGB sensor. This means that the camera captures both events and RGB images, and that both modalities are spatially aligned. The ground truth is obtained using a 16-channel Velodyne Puck LITE LiDAR.



(a) Hexacopter Indoor Flight with Vicon Motion Capture.

(b) Hexacopter Outdoor Flight with Qualisys Motion Capture.

(c) Handheld with Difficult Lighting Conditions.

Figure 2.24: Example samples from the MVSEC dataset, adapted from [44]. The figure shows event data overlaid on grayscale intensity images across indoor, outdoor, handheld, daytime, and nighttime scenarios. This illustrates the diversity of MVSEC, while the lower spatial resolution compared to DSEC is an important dataset characteristic for this thesis.

3

Scientific Paper

Learning Event Representations for Vision Foundation Models for Monocular Depth Estimation

Maosheng Jiang Hesam Araghi Nergis Tömen
Pattern Recognition & Bioinformatics Group, Delft University of Technology, The Netherlands
{m.jiang-5, H.Araghi, N.Tomen}@tudelft.nl
https://github.com/maotek/MSCThesis

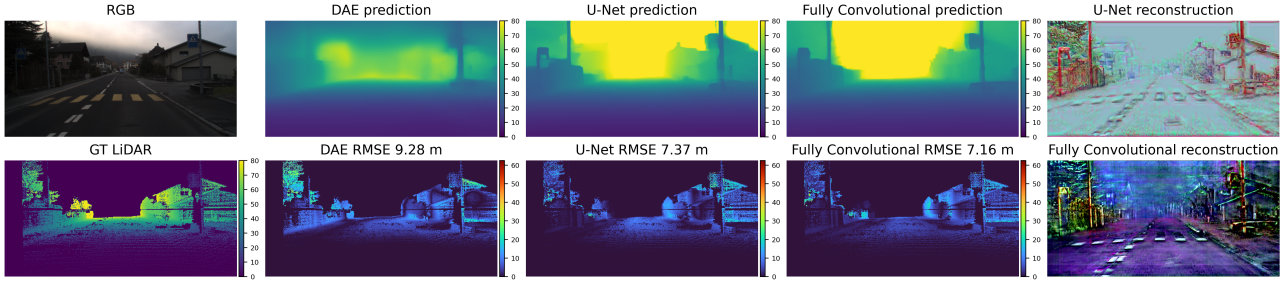


Figure 1. Qualitative comparison on DSEC for event-based monocular depth estimation. Depth AnyEvent (DAE) [1] is the baseline method, while U-Net DAv2 and FullyConv DAv2 are our proposed methods. The top row shows the RGB reference image, the predicted depth maps from DAE, U-Net DAv2, and FullyConv DAv2, and the event representation learned by the U-Net representation learner. The bottom row shows the sparse LiDAR ground truth, the masked RMSE error maps for each prediction, and the event representation learned by the FullyConv representation learner. Our learned representations produce sharper depth predictions and lower errors than DAE, showing that learning the event representation improves in-domain performance on DSEC.

Abstract

Event cameras are a novel sensing modality, but the lack of densely annotated datasets remains a major limitation for tasks such as monocular depth estimation. To address this, we investigate how Vision Foundation Models (VFMs), trained on large-scale RGB datasets, can be leveraged for event-based depth estimation. Previous work combines handcrafted event representations with fine-tuning of VFMs to adapt them to the event domain. In contrast, we learn an event representation while keeping the VFM frozen. We evaluate two representation learners, a U-Net and a Fully Convolutional (FullyConv) model, on DSEC and MVSEC. The results show that learned event representations are highly effective in-domain: both models outperform all baselines on DSEC, including Depth AnyEvent (DAE) and direct RGB input to Depth Anything V2 (DAv2), while the FullyConv model remains competitive on MVSEC. Cross-dataset experiments show that this improvement does not consistently transfer under domain shift. These findings indicate that learning the input representation is a strong strategy for in-domain event-based depth estimation, but that representation learning alone is not sufficient to guarantee robust cross-dataset generalization.

1. Introduction

Depth estimation is a fundamental topic within computer vision. It has several applications within the automotive [26], robotics [7, 31], and AR/VR [9] domains. Neuromorphic cameras, also known as event cameras, are a novel type of sensor inspired by the biological principles of the human eye. This sensing modality holds promise in the computer vision field due to the higher temporal resolution, higher dynamic range, and lower power consumption compared with conventional RGB cameras. These advantages make event cameras promising for depth estimation.

However, due to their novelty, datasets with dense depth ground truth remain scarce. To address this limitation, we aim to leverage VFMs, which are trained on large-scale RGB datasets, for depth estimation in the event domain. A key challenge is that event cameras operate fundamentally differently from RGB cameras. While RGB cameras synchronously capture images at a fixed frame rate, event cameras operate asynchronously, where each pixel indepen-

dently generates events indicating whether the intensity increased or decreased.

Previous work addresses this gap by utilizing a fixed event representation and subsequently fine-tuning VFMs to adapt them to the event domain for depth estimation [1, 33]. However, this approach has several limitations. First, a fixed event representation may be suboptimal for the VFM, even after fine-tuning. Furthermore, fine-tuning VFMs can reduce generalizability across datasets, as the model becomes more specialized to the training data. Lastly, fine-tuning VFMs requires significant computational resources.

In this work, we propose a methodology to adapt RGB-based VFMs to the event domain for monocular depth estimation by learning the event representation while keeping the VFM frozen. Our results show that this strategy is highly effective in-domain on DSEC and remains competitive on MVSEC. However, cross-dataset experiments show that freezing the VFM does not consistently improve generalization under domain shift, indicating that the learned representation itself can still become dataset-specific.

Our contributions are as follows:

- We introduce an event-to-VFM adaptation strategy that learns an event representation while keeping the RGB-based VFM frozen.
- We compare two architectures for learning the event representation: a U-Net-based architecture and a FullyConv architecture.
- We show that learned event representations achieve strong in-domain performance, especially on DSEC, while cross-dataset experiments reveal that representation learning alone does not consistently improve generalization.
- We analyze the trade-off between accuracy and computational efficiency, showing that U-Net provides the strongest runtime trade-off, whereas FullyConv achieves competitive accuracy with substantially fewer trainable parameters.

2. Related Work

Image-based Depth Estimation. Monocular depth estimation has seen significant progress over the past decade, initially driven by Convolutional Neural Network (CNN) architectures such as the one proposed by Godard et al. [12].

Building on these foundations, the introduction of the Vision Transformer (ViT) [8] enabled models to capture long-range dependencies more effectively, leading to further improvements in downstream vision tasks. In the context of depth estimation, transformer-based approaches such as those proposed by Ranftl et al. [22, 23] have demonstrated improved accuracy and generalization compared to earlier CNN-based methods in the RGB domain.

Vision Foundation Models. Building upon the DPT framework [23], a recent line of VFMs for depth estimation has emerged, notably the Depth Anything models [28]. These models combine DPT-based architectures with large-scale training and improved feature aggregation strategies. DPT leverages a hybrid encoder-decoder design that extracts multi-resolution features from intermediate transformer layers and fuses them through a convolutional decoder.

Subsequent work, such as Depth Anything V2 (DAv2) [29], further improves performance and generalization. These models are trained on large-scale RGB image datasets and demonstrate strong zero-shot generalization across a wide range of domains. Other notable models include Depth Pro [2], MiDaS [22], and more recent developments such as Depth Anything V3 [17].

Event-based Depth Estimation. Depth estimation with event cameras has also seen recent developments. A line of work focuses on estimating depth using only event data [13, 15, 19, 20]. In these approaches, the asynchronous event stream is first converted into a voxel grid representation, after which standard neural networks are applied to predict depth.

Other approaches incorporate sensor fusion with additional sensors such as LiDAR to improve accuracy. These methods leverage the complementary nature of LiDAR, which provides sparse but accurate depth measurements [3, 4].

Another line of work explores multi-modal fusion of event data with RGB images. In these approaches, event data is used to complement RGB images, providing additional temporal and structural information that improves depth estimation accuracy [6, 16, 18].

RGB-to-Event Distillation. Since event data has a fundamentally different modality compared to RGB images, existing RGB-based depth estimation models do not directly generalize to the event domain. To address this modality gap, a line of work has emerged that focuses on adapting pretrained RGB models to event-based data.

These approaches typically leverage powerful Vision Foundation Models trained on large-scale RGB datasets as teacher models, while an event-based model acts as the student. In this setting, knowledge is transferred from the RGB domain to the event domain through a distillation process.

Zhang et al. [30] introduce a cross-modality knowledge distillation framework in which a pretrained RGB-based teacher model [21] supervises an event-based spiking neural network as the student. The teacher generates depth maps from RGB images, which are then used as pseudo ground truth to train the student model on temporally aligned event data.

Other methods [1, 16, 33] follow a similar paradigm, using DAv2 [29] as the teacher model to guide event-based depth estimation.

In particular, our work is inspired by Bartolomei et

al. [1], who propose Depth AnyEvent (DAE), a distillation-based approach for event-based depth estimation. Their method converts events into a Tencode representation [14], which is then used as input to a Vision Foundation Model that is subsequently fine-tuned for depth estimation.

However, using fixed event representations such as Tencode, as in DAE, or voxel grids, as in Depth Any Event Stream [33], may be suboptimal for the VFMs they are paired with. Furthermore, fine-tuning the VFM on these representations can lead to reduced generalizability across datasets, as the model becomes more specialized to the training data.

This motivates the core idea of our work: instead of relying on a fixed event representation, we propose to learn the representation jointly with the model while keeping the VFM frozen.

3. Proposed Method

Our goal is to bridge the modality gap between event data and RGB-based VFMs without modifying the VFM itself. Instead of relying on a handcrafted event representation such as Tencode [14], we introduce a small neural network that learns to convert event data into a representation that can be processed by a frame-based monocular depth estimation model such as DAv2 [29].

Concretely, we place a lightweight representation learner in front of a pretrained VFM. This network takes a slice of event data as input and produces an image-like representation that is compatible with the underlying VFM. The VFM is kept frozen during training, while only the representation learner is optimized. This allows us to retain the strong generalization capabilities of the VFM while avoiding the computational cost of fine-tuning.

We investigate two different architectures for this representation learner:

- A U-Net-based architecture [25], which captures multi-scale spatial features through an encoder-decoder structure with skip connections.
- A Fully Convolutional network architecture [5], which serves as a lightweight alternative.

Both architectures are trained end-to-end together with a frozen DAv2 as the VFM, allowing the representation learner to adapt the event input to the feature space expected by the DAv2.

3.1. U-Net-Based Representation Learner

The first representation learner architecture is based on the U-Net [25]. It consists of a U-Net with two encoder blocks and two decoder blocks, followed by a 1×1 convolution to map the output to a 3-channel tensor, making it compatible with DAv2. Finally, a sigmoid activation is applied to produce the final event representation.

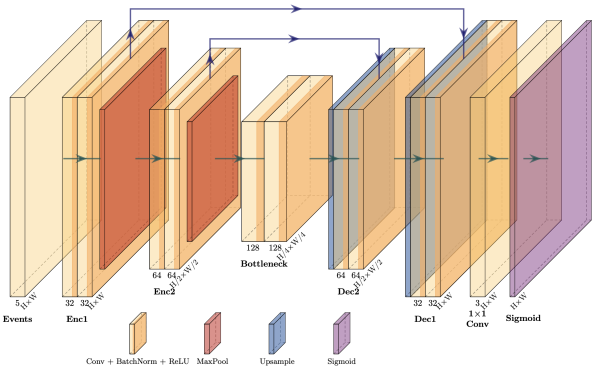


Figure 2. U-Net-based representation learner used to adapt event data to the frozen DAv2 backbone. The network encodes the voxel-grid event input through two downsampling stages and produces a three-channel image-like representation using decoder blocks with skip connections.

Each block consists of two convolutional layers, each followed by batch normalization and a ReLU activation. The base number of channels is set to 32, and is doubled after each encoder stage.

The architecture of this representation learner is shown in Figure 2.

3.2. Fully Convolutional Representation Learner

The second representation learner architecture is based on a Fully Convolutional network (FullyConv) [5]. This network differs from the U-Net-based architecture in that it does not perform any downsampling, and therefore maintains the spatial resolution throughout the network.

We adopt the architecture proposed by the original authors. It consists of 9 convolutional blocks, followed by a final 1×1 convolution to map the features to a 3-channel output, making it compatible with the DAv2 input.

Each block consists of a dilated convolution, a learnable normalization layer, and a leaky ReLU activation. The use of dilated convolutions allows the network to capture a larger receptive field without reducing spatial resolution.

The architecture of this representation learner is shown in Figure 3.

3.3. Training Details

The final monocular depth estimation pipeline consists of the representation learner, either U-Net or FullyConv, followed by a pretrained DAv2 model with a ViT-S backbone. The overall pipeline is illustrated in Figure 4.

During training, the DAv2 model is kept frozen, and only the representation learner is optimized. Concretely, gradients are not propagated through the DAv2 parameters, ensuring that its pretrained weights remain unchanged. The

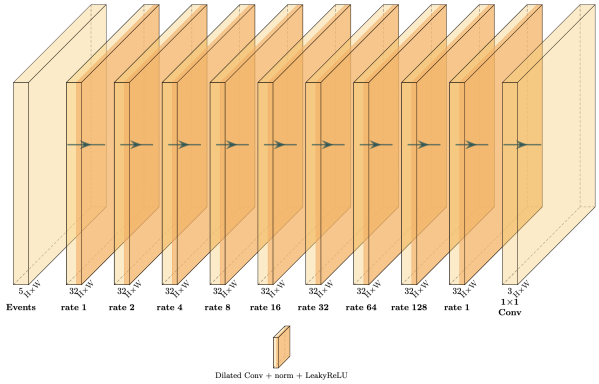


Figure 3. FullyConv representation learner used to convert voxel-grid event input into a three-channel DAv2-compatible representation. The model uses a sequence of dilated convolutional blocks, with the dilation rate indicated for each layer, and preserves the spatial resolution throughout the network. [5]

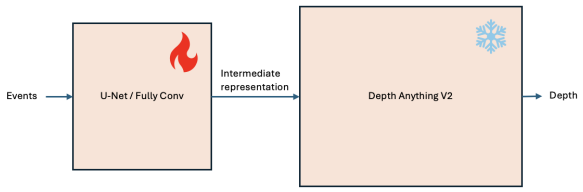


Figure 4. Overview of the proposed event-to-depth pipeline. A trainable representation learner converts voxel-grid event input into a three-channel representation, which is then processed by a frozen DAv2 backbone to predict monocular depth. By optimizing only the representation learner, the method adapts event data to the VFM while preserving the pretrained backbone.

representation learner is optimized using the Adam optimizer.

3.3.1. Event Input Representation

For all experiments, we construct a fixed temporal slice of the event stream using a time window ΔT. Each event is represented as (x, y, t, p), where (x, y) denotes the pixel location, t the timestamp, and $p \in \{0, 1\}$ the polarity.

Within each time window, the events are aggregated into a voxel grid representation with B temporal bins. Specifically, the timestamps are normalized to the range $[0, B-1]$ and discretized into bins, resulting in a tensor of size $B \times H \times W$. This representation provides a dense and temporally structured input that is compatible with convolutional processing.

3.3.2. Loss Function

We adopt the same loss formulation as used in DAE [1]. The loss consists of a scale- and shift-invariant loss and a gradient regularization term [22].

The scale- and shift-invariant loss is defined as:

$$\mathcal{L}_{si}(\hat{D}, D^*) = \frac{1}{2|\mathcal{M}|} \sum_{(x,y)\in\mathcal{M}} \Big(\hat{D}(x,y) - D^*(x,y)\Big)^2 \tag{1}$$

Here, $\mathcal{M}$ denotes the set of valid pixels. D^* denotes the ground truth depth, and $\hat{D}$ is the scaled and shifted version of the predicted depth D, defined as:

$$\hat{D} = sD + t \tag{2}$$

The scale s and shift t are computed using a least-squares formulation:

$$(s, t) = \arg\min_{s,t} \sum_{(x,y)\in\mathcal{M}} \big(sD(x,y) + t - D^*(x,y)\big)^2 \tag{3}$$

This formulation ensures that the loss is invariant to global scale and shift differences between the prediction and the ground truth, which is important since monocular depth estimation suffers from scale ambiguity.

In addition, a multi-scale gradient regularization term is applied to encourage spatial smoothness while preserving edges:

$$\mathcal{L}_{reg}(\hat{D}, D^*) = \sum_{k=1}^{K} \frac{1}{|\mathcal{M}_k|} \sum_{(x,y)\in\mathcal{M}_k} \big(|\nabla_x R_k| + |\nabla_y R_k|\big) \tag{4}$$

where $R_k = \hat{D}_k - D_k^*$ represents the difference map at scale k.

The final loss is given by the sum of the two terms:

$$\mathcal{L} = \mathcal{L}_{si} + \lambda \mathcal{L}_{reg} \tag{5}$$

3.4. Processing Details

One important detail is that DAv2 predicts relative inverse depth by default. As a result, the output is not directly comparable to the metric depth ground truth used in our experiments.

To address this, we invert the predicted inverse depth to obtain a depth estimate:

$$D = \frac{1}{D_{\text{DAv2}} + 1} \tag{6}$$

where D_{DAv2} denotes the inverse depth predicted by DAv2.

We found that using a very small constant (e.g., $1e^{-6}$) led to unstable results due to large depth values when D_{DAv2} approaches zero. In contrast, adding a constant offset of 1 results in more stable behavior during training. This behavior is shown in Figure 5.

4. Experiments

In this section, we describe the experimental settings, datasets, evaluation metrics, and present the experimental results along with their discussion.

4.1. Experimental Settings

Hyperparameters

For all experiments, we set the time window ΔT to 50 ms and the number of temporal bins B to 5. The gradient regularization weight λ is set to 0.25, following [1].

We train all models using the Adam optimizer with a batch size of 10 and a learning rate of $1e^{-4}$ without weight decay. All experiments are implemented in PyTorch and trained on a single NVIDIA A40 GPU with 48 GB VRAM.

All networks are trained for 50 epochs. Due to increased computational cost and memory usage, RGB-input-based models are trained for 25 epochs.

For all training experiments, we use the ViT-S variant of DAv2. Furthermore, since DAv2 outputs relative inverse depth, we set the inversion constant to 1 as defined in Eq. (6).

4.2. Datasets and Evaluation

We use two event-based datasets for our experiments: DSEC [11] and MVSEC [32].

The DSEC dataset consists of 53 driving sequences recorded under varying illumination conditions. Events are captured using two stereo Prophesee Gen3.1 event cameras with a resolution of 640×480. In addition, the dataset provides RGB images from two separate stereo FLIR Blackfly cameras at a resolution of 1440×1080. Ground truth depth is obtained using a 16-channel Velodyne VLP-16 LiDAR. In total, DSEC contains approximately 26k samples, which we split following [1] into 19k training samples and 7k testing samples, both consisting of a mix of daytime and nighttime sequences.

The MVSEC dataset provides event data across multiple scenarios, including motorcycles, cars, and drones. In this work, we focus on the car driving sequences. Events are recorded using two stereo DAVIS 346B event cameras with a resolution of 346×260. Unlike DSEC, the DAVIS sensors include an integrated RGB camera, meaning that event and RGB data are spatially aligned by design. In contrast, DSEC uses separately mounted event and RGB cameras, requiring additional alignment. Ground truth in MVSEC is obtained using a 16-channel Velodyne Puck LITE LiDAR. In total, we split the dataset into 12k training samples and 20k testing samples. The training set consists of a single daytime sequence, while the testing set includes one daytime and three nighttime sequences.

4.2.1. Evaluation Metrics

For the evaluation, we use several standard depth estimation metrics: absolute relative error (Abs Rel), squared relative error (Sq Rel), root mean squared error (RMSE), logarithmic RMSE (RMSE log), and scale-invariant logarithmic error (SI log). In addition, we report accuracy under different thresholds, defined as $\delta < 1.25$, $\delta < 1.25^2$, and $\delta < 1.25^3$.

During evaluation on the test sets, we apply a scale and shift to the predicted depth maps to align them with the ground truth before computing the metrics.

4.3. In-Domain Evaluation

In this experiment, we evaluate both models, U-Net and FullyConv, when trained and evaluated on the same dataset, DSEC or MVSEC. The results are shown in Table 1. In Figure 6, we show qualitative visualizations of our methods on DSEC and MVSEC.

For DSEC, we use a batch size of 10 for both the U-Net and FullyConv models. For the U-Net, we apply a 320×640 center crop as the only preprocessing step. For the FullyConv model, we instead train with a 260×346 random crop. This crop size matches the MVSEC resolution and is used to keep the same checkpoint consistent with the cross-dataset generalization experiment discussed in the next section. When validating this model on DSEC, we use a 320×640 center crop as the only preprocessing step.

For MVSEC, we also use a batch size of 10 for both models. In this case, we apply a 260×346 crop as the only preprocessing step.

We compare our method against five different baselines.

First, we directly feed the Tencode representation into pretrained DAv2 without any fine-tuning.

Second, we feed the RGB images from the datasets directly into DAv2 without any fine-tuning. This serves as a baseline using the input modality originally expected by DAv2.

Third, we compare against DAE [1] using their pretrained weights. Figure 1 shows a qualitative comparison of DAE against our methods for the DSEC dataset.

Fourth, we evaluate an event-based depth estimation pipeline consisting of E2VID [24] as an event-to-intensity reconstruction method. We use pretrained E2VID weights to reconstruct intensity images from the event data and subsequently feed these reconstructed images into DAv2. Since the reconstructed intensity images are single-channel, we duplicate them to three channels before passing them to DAv2.

Finally, we evaluate a similar reconstruction-based pipeline using ET-Net [27] instead of E2VID.

The results in Table 1 show that learned event representations are highly effective on DSEC. Both representation learners outperform all baselines across all metrics. Compared to DAE, U-Net DAv2 reduces Abs Rel from 0.201

Table 1. In-domain depth estimation results on DSEC and MVSEC. We compare event-to-depth baselines, RGB and event reconstruction pipelines with DAv2, and our trainable event-to-DAv2 adapters using standard depth estimation metrics. Best results are shown in bold and second-best results are underlined. Our learned representations achieve the best overall performance on DSEC, while FullyConv DAv2 remains competitive with DAE on MVSEC.

Model	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	SI log ↓	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
DSEC								
Depth AnyEvent (DAE) [1]	0.201	0.079	8.880	0.266	0.077	0.664	0.917	0.975
Tencode DAv2	0.260	0.160	10.349	0.312	0.106	0.588	0.860	0.952
RGB DAv2	0.170	0.069	7.748	0.225	0.053	0.761	0.944	0.983
E2VID DAv2 [24]	0.214	0.112	9.160	0.278	0.084	0.674	0.900	0.967
ETNet DAv2 [27]	0.236	0.131	9.723	0.299	0.098	0.630	0.881	0.960
U-Net DAv2 (Ours)	**0.148**	**0.049**	**7.109**	**0.196**	**0.042**	**0.799**	**0.959**	**0.989**
FullyConv DAv2 (Ours)	0.149	0.049	7.128	0.200	0.044	0.793	0.956	0.989
MVSEC								
Depth AnyEvent (DAE) [1]	0.377	0.741	**6.647**	0.454	0.226	0.469	0.745	0.882
Tencode DAv2	0.622	1.230	9.111	0.630	0.385	0.326	0.569	0.737
RGB DAv2	0.421	0.949	7.254	0.522	0.278	0.454	0.748	0.869
E2VID DAv2 [24]	0.667	1.379	9.481	0.650	0.399	0.298	0.548	0.722
ETNet DAv2 [27]	0.763	1.731	9.751	0.695	0.448	0.271	0.504	0.680
U-Net DAv2 (Ours)	0.427	0.785	6.989	0.536	0.305	0.413	0.691	0.846
FullyConv DAv2 (Ours)	**0.375**	**0.736**	6.673	**0.421**	**0.180**	**0.472**	**0.752**	**0.892**

Table 2. Cross-dataset generalization results between DSEC and MVSEC. Models are trained on one dataset and evaluated on the other without additional fine-tuning, allowing evaluation under domain shift. Best results are shown in bold and second-best results are underlined. FullyConv DAv2 is competitive when transferring from DSEC-to-MVSEC, but neither representation learner consistently outperforms DAE across both transfer directions.

Model	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	SI log ↓	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
DSEC → MVSEC								
Depth AnyEvent (DAE)	0.471	**0.875**	8.013	0.607	0.386	0.392	0.655	0.807
U-Net DAv2	0.506	0.977	8.197	0.598	0.363	0.383	0.645	0.799
FullyConv DAv2	**0.448**	0.875	**7.668**	**0.573**	**0.342**	**0.431**	**0.690**	**0.828**
MVSEC → DSEC								
Depth AnyEvent (DAE)	**0.280**	**0.158**	**10.953**	**0.317**	**0.101**	**0.533**	**0.842**	**0.955**
U-Net DAv2	0.307	0.181	11.761	0.361	0.133	0.477	0.803	0.940
FullyConv DAv2	0.309	0.216	11.721	0.362	0.137	0.511	0.808	0.933

to 0.148 and RMSE from 8.880 to 7.109, corresponding to relative reductions of approximately 26% and 20%, respectively. FullyConv DAv2 achieves similar performance, with an Abs Rel of 0.149 and RMSE of 7.128. Both variants also outperform the RGB DAv2 baseline, despite using event data rather than the RGB input modality for which DAv2 was originally trained. This suggests that the learned representations are not merely image-like reconstructions, but are adapted to the input distribution expected by the frozen DAv2 backbone.

On MVSEC, the results are more mixed. FullyConv DAv2 obtains the best Abs Rel among the compared methods, slightly improving over DAE from 0.377 to 0.375. It also improves SI log from 0.226 to 0.180 and improves the threshold accuracies, with $\delta < 1.25^3$ increasing from 0.882 to 0.892. However, DAE remains slightly better in RMSE,

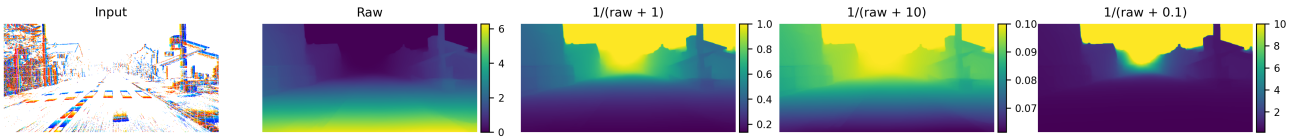


Figure 5. Effect of the inversion constant when converting the relative inverse depth predicted by DAv2 into depth estimates. The raw relative inverse depth prediction is shown together with the resulting depth maps after applying $1/(D_{\mathrm{DAv2}} + c)$ for different constants (c). Small constants produce unstable depth estimates for nearby objects, whereas larger constants compress close-range depth variation and lead to more stable training behavior.

with 6.647 compared to 6.673 for FullyConv DAv2, and U-Net DAv2 performs worse than both methods. Therefore, the MVSEC results should be interpreted as competitive rather than clearly superior.

The difference between DSEC and MVSEC is likely related to dataset characteristics. DSEC has higher spatial resolution and a more diverse training split, while MVSEC has lower resolution and a training set consisting of a single daytime sequence. In addition, the MVSEC test split contains both daytime and nighttime sequences, which introduces a larger train-test domain shift. These factors make it harder for the representation learner to learn a robust mapping from events to the frozen DAv2 input space.

Overall, the in-domain results support the main hypothesis that a representation learner can adapt event data to a frozen RGB-based VFM. However, the strength of this adaptation depends on the dataset. The approach is clearly beneficial on DSEC, while on MVSEC the improvement is limited and architecture-dependent.

4.4. Cross-Dataset Generalization

Table 2 shows the generalizability of our method across the DSEC and MVSEC datasets. In this experiment, we evaluate how well the models transfer across datasets by training on one dataset and evaluating on the other. Specifically, we train on DSEC and evaluate on MVSEC, and vice versa.

For DAE [1], we follow the same protocol by taking the pretrained checkpoint trained on one dataset (e.g. DSEC) and evaluating it on the other dataset (e.g. MVSEC), without any additional fine-tuning.

This setup enables us to assess the robustness of the learned representations across different domains, as DSEC and MVSEC differ in camera properties, environments, and data distributions. In particular, we are interested in evaluating the generalizability of our method, as we keep DAv2 frozen during training, which may help the model generalize across datasets. In contrast, DAE fine-tunes the backbone, which may lead to stronger adaptation to the training dataset but potentially reduced generalization across datasets.

The results in Table 2 show that learning the event representation does not consistently improve cross-dataset generalization. In the DSEC to MVSEC setting, FullyConv DAv2 improves over DAE on most metrics. Abs Rel decreases from 0.471 to 0.448, RMSE decreases from 8.013 to 7.668, SI log decreases from 0.386 to 0.342, and $\delta < 1.25$ increases from 0.392 to 0.431. This indicates that the FullyConv representation transfers reasonably well from DSEC to MVSEC. However, the U-Net variant performs worse than DAE in the same transfer direction, with Abs Rel increasing from 0.471 to 0.506 and RMSE increasing from 8.013 to 8.197. Therefore, the benefit is architecture-dependent rather than a general effect of freezing DAv2.

In the MVSEC to DSEC setting, both learned representation models underperform DAE. Depth AnyEvent achieves an Abs Rel of 0.280 and RMSE of 10.953, whereas U-Net DAv2 obtains 0.307 and 11.761, and FullyConv DAv2 obtains 0.309 and 11.721. This shows that representations learned from MVSEC do not transfer as effectively to DSEC. The direction of transfer is therefore important: training on the higher-resolution and more diverse DSEC dataset transfers better to MVSEC than training on MVSEC transfers to DSEC.

A likely explanation is that DAE benefits from additional pretraining on the synthetic EventScape dataset [10], which provides a more diverse training distribution before fine-tuning on the target event dataset. In contrast, our method trains the representation learner only on the source dataset while keeping DAv2 frozen. Although this preserves the pretrained RGB-depth backbone, the representation learner can still learn dataset-specific event statistics. This effect is especially relevant for MVSEC, where the training data is lower resolution and less diverse.

Overall, the cross-dataset experiments refine the main conclusion of this work. Freezing DAv2 is not sufficient by itself to guarantee better generalization. The learned representation must also be domain-invariant. In our experiments, representation learning improves in-domain performance, especially on DSEC, but does not consistently solve the domain shift between DSEC and MVSEC.

4.5. In-Domain Ablation Studies

In this section, we analyze the impact of several design choices in our method. We study the effect of using a grayscale representation, changing the output activation, modifying the U-Net architecture, and making the inver-

Table 3. In-domain ablation study of the proposed representation learners on DSEC and MVSEC. We evaluate output dimensionality, U-Net capacity, output activation, and the use of a learnable inverse-depth constant. Best and second-best results are marked in bold and underlined within each model family and dataset. Learnable inversion constants generally improve performance, while reduced U-Net capacity is mainly beneficial on MVSEC.

Model	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	SI log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
DSEC								
U-Net DAv2 (baseline)	0.148	0.049	7.109	0.196	0.042	0.799	0.959	0.989
U-Net DAv2 (ReLU activation)	0.149	0.050	7.159	0.195	0.041	0.797	0.959	0.990
U-Net DAv2 (grayscale)	0.157	0.055	7.389	0.205	0.046	0.780	0.952	0.988
U-Net DAv2 (16 channels)	0.158	0.053	7.517	0.205	0.046	0.777	0.952	0.988
U-Net DAv2 (1 encoder, 1 decoder)	0.166	0.058	7.769	0.218	0.051	0.758	0.945	0.986
U-Net DAv2 (learnable constant)	**0.144**	**0.047**	**7.053**	**0.185**	**0.037**	**0.810**	**0.963**	**0.991**
FullyConv DAv2 (baseline)	0.149	0.049	7.128	0.200	0.044	0.793	0.956	0.989
FullyConv DAv2 (grayscale)	0.159	0.053	7.528	0.209	0.048	0.775	0.951	0.987
FullyConv DAv2 (learnable constant)	**0.143**	**0.046**	**7.018**	**0.183**	**0.037**	**0.811**	**0.963**	**0.992**
MVSEC								
U-Net DAv2 (baseline)	0.427	0.785	6.989	0.536	0.305	0.413	0.691	0.846
U-Net DAv2 (ReLU activation)	0.427	0.766	7.129	0.540	0.306	0.408	0.690	0.848
U-Net DAv2 (grayscale)	0.450	0.807	7.209	0.562	0.329	0.384	0.667	0.833
U-Net DAv2 (16 channels)	0.395	**0.702**	6.841	**0.481**	**0.249**	0.441	**0.726**	**0.872**
U-Net DAv2 (1 encoder, 1 decoder)	0.430	0.800	7.176	0.532	0.301	0.416	0.695	0.846
U-Net DAv2 (learnable constant)	**0.391**	0.716	**6.795**	0.504	0.273	**0.443**	0.724	0.869
FullyConv DAv2 (baseline)	0.375	0.736	6.673	0.421	0.180	0.472	0.752	0.892
FullyConv DAv2 (grayscale)	0.376	0.726	6.675	0.426	0.184	0.465	0.751	0.894
FullyConv DAv2 (learnable constant)	**0.362**	**0.713**	**6.501**	**0.410**	**0.172**	**0.482**	**0.764**	**0.901**

sion constant learnable. These ablations allow us to better understand which components contribute to the final performance. The results are shown in Table 3.

4.5.1. Grayscale Representation

In this experiment, we evaluate whether a single-channel representation is sufficient, or whether the representation learner benefits from producing a three-channel output. The grayscale variant outputs one channel, which is duplicated to three channels before being passed to DAv2, since the VFM requires a three-channel input.

The results show that reducing the output to a grayscale representation generally decreases performance. On DSEC, the effect is clear for both architectures. For U-Net, Abs Rel increases from 0.148 to 0.157 and RMSE increases from 7.109 to 7.389. For FullyConv, Abs Rel increases from 0.149 to 0.159 and RMSE increases from 7.128 to 7.528. This indicates that the three output channels are not redundant. Instead, the representation learner uses the channel dimension to encode event information in a form that is more compatible with the RGB-like input space expected by DAv2.

On MVSEC, the effect is smaller for FullyConv, where the grayscale variant remains close to the three-channel baseline. However, for U-Net, the grayscale representation still reduces performance. Overall, the three-channel representation is preferable for in-domain performance, especially on DSEC.

4.5.2. Output Activation

In this experiment, we evaluate the effect of replacing the sigmoid output activation of the U-Net representation learner with a ReLU activation. The sigmoid constrains the learned representation to the range $[0, 1]$, which is closer to a normalized image-like input. In contrast, ReLU only enforces non-negative outputs and does not impose an upper bound.

The results show that the ReLU activation performs similarly to the sigmoid baseline. On DSEC, the differences are small: Abs Rel changes from 0.148 to 0.149, while RMSE changes from 7.109 to 7.159. On MVSEC, Abs Rel remains unchanged at 0.427, while RMSE slightly increases from 6.989 to 7.129. These results indicate that the output activation is not the main factor controlling in-domain performance. We therefore keep the sigmoid activation as the default choice, since it produces bounded image-like repre-

Table 4. Cross-dataset generalization ablation study of the proposed representation learners. Models are trained on the source dataset and evaluated on the target dataset without fine-tuning while varying output dimensionality, U-Net capacity, output activation, and the inverse-depth constant. Best and second-best results are marked in bold and underlined within each model family and transfer direction. The results show that design choices improving in-domain performance do not necessarily improve cross-dataset generalization.

Model	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	SI log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
DSEC → MVSEC								
U-Net DAv2 (baseline)	<u>0.506</u>	<u>0.977</u>	**8.197**	0.598	0.363	<u>0.383</u>	<u>0.645</u>	<u>0.799</u>
U-Net DAv2 (ReLU activation)	0.542	1.073	8.446	0.610	0.371	0.366	0.624	0.781
U-Net DAv2 (grayscale)	0.600	1.242	8.919	0.629	0.388	0.335	0.589	0.756
U-Net DAv2 (16 channels)	**0.496**	**0.966**	<u>8.259</u>	**0.584**	**0.345**	**0.385**	**0.657**	**0.810**
U-Net DAv2 (1 encoder, 1 decoder)	0.530	1.046	8.560	0.595	0.352	0.368	0.631	0.791
U-Net DAv2 (learnable constant)	0.519	0.998	8.267	<u>0.585</u>	<u>0.348</u>	0.379	0.638	0.794
FullyConv DAv2 (baseline)	**0.448**	<u>0.875</u>	**7.668**	**0.573**	**0.342**	**0.431**	**0.690**	**0.828**
FullyConv DAv2 (grayscale)	0.515	0.939	8.329	0.607	0.368	0.352	0.628	0.799
FullyConv DAv2 (learnable constant)	<u>0.463</u>	**0.865**	<u>7.835</u>	<u>0.580</u>	<u>0.351</u>	<u>0.402</u>	<u>0.670</u>	<u>0.820</u>
MVSEC → DSEC								
U-Net DAv2 (baseline)	0.307	0.181	11.761	0.361	0.133	0.477	0.803	0.940
U-Net DAv2 (ReLU activation)	0.255	0.128	10.498	0.306	0.098	0.565	0.869	0.961
U-Net DAv2 (grayscale)	0.294	0.178	11.236	0.351	0.129	0.515	0.821	0.941
U-Net DAv2 (16 channels)	<u>0.251</u>	<u>0.125</u>	<u>10.403</u>	<u>0.294</u>	<u>0.091</u>	<u>0.578</u>	<u>0.874</u>	<u>0.963</u>
U-Net DAv2 (1 encoder, 1 decoder)	**0.229**	**0.108**	**9.747**	**0.280**	**0.084**	**0.626**	**0.894**	**0.968**
U-Net DAv2 (learnable constant)	0.297	0.178	11.605	0.340	0.119	0.506	0.818	0.944
FullyConv DAv2 (baseline)	0.309	0.216	11.721	<u>0.362</u>	<u>0.137</u>	0.511	0.808	0.933
FullyConv DAv2 (grayscale)	<u>0.300</u>	<u>0.213</u>	<u>11.692</u>	0.366	0.140	<u>0.534</u>	<u>0.815</u>	<u>0.934</u>
FullyConv DAv2 (learnable constant)	**0.289**	**0.186**	**11.392**	**0.337**	**0.120**	**0.535**	**0.828**	**0.944**

sentations and performs consistently across datasets.

4.5.3. U-Net Architecture Variants

In this experiment, we analyze the effect of reducing the capacity of the U-Net representation learner. We consider two variants. First, we reduce the number of base channels from 32 to 16, which reduces the width of the model. Second, we use a shallower architecture with one encoder block and one decoder block, which reduces the depth of the model.

The results show different trends for DSEC and MVSEC. On DSEC, reducing the U-Net capacity decreases performance. The 16-channel variant increases Abs Rel from 0.148 to 0.158, while the shallower one-encoder/one-decoder variant further increases Abs Rel to 0.166. RMSE follows the same trend, increasing from 7.109 to 7.517 and 7.769, respectively. This suggests that the default U-Net has useful capacity for learning event representations on DSEC.

On MVSEC, the 16-channel variant improves over the default U-Net, reducing Abs Rel from 0.427 to 0.395 and RMSE from 6.989 to 6.841. This suggests that a smaller model can be beneficial when the dataset is lower resolution or more prone to overfitting. However, the shallower one-encoder/one-decoder variant does not show the same improvement, indicating that reducing the depth too much limits the representation learner. Therefore, the optimal U-Net capacity is dataset-dependent: DSEC benefits from the default capacity, while MVSEC benefits from a narrower but not overly shallow model.

4.5.4. Learnable Inversion Constant

In this experiment, we evaluate the effect of making the inversion constant learnable instead of fixed. The output of DAv2 represents relative inverse depth, which we convert to relative depth using an inversion of the form $1/(d_{\text{pred}} + c)$, where c is a constant. Instead of using a fixed value for c, we make it a learnable parameter and optimize it jointly with the representation learner.

The learnable inversion constant improves performance on DSEC for both U-Net and FullyConv. For U-Net, Abs Rel decreases from 0.148 to 0.144, RMSE decreases from 7.109 to 7.053, and SI log decreases from 0.042 to 0.037. For FullyConv, Abs Rel decreases from 0.149 to 0.143, RMSE decreases from 7.128 to 7.018, and SI log decreases from 0.044 to 0.037. This indicates that the fixed inversion constant is not always optimal, and that learning this parameter helps align the relative inverse-depth output of DAv2

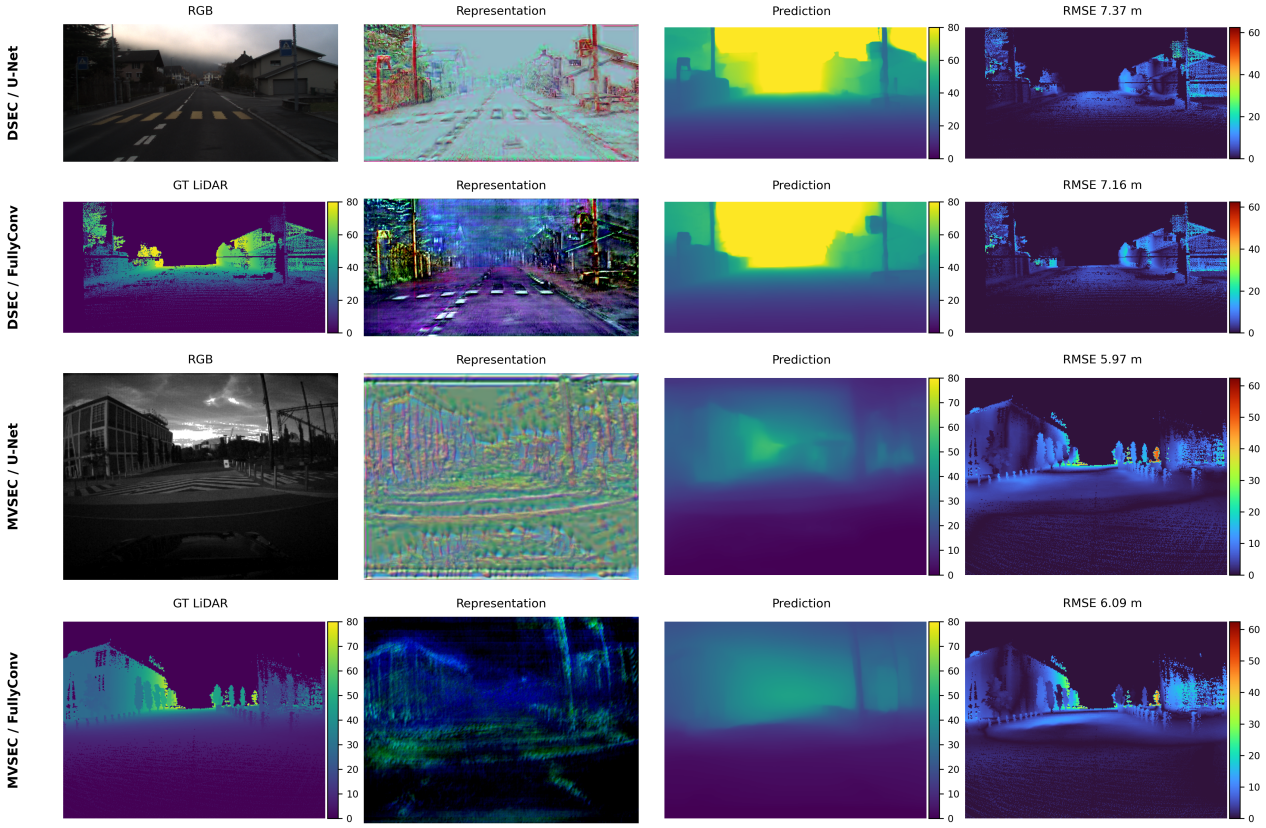


Figure 6. Qualitative results of the U-Net and FullyConv representation learners on DSEC and MVSEC. For each dataset and model, the figure shows the reference input, learned event representation, predicted depth, and masked RMSE map. Both representation learners recover plausible scene geometry, with sharper depth predictions on DSEC than on MVSEC. The lower visual quality of the learned MVSEC representations suggests that the representation learners are more affected by the lower resolution and noisier ground truth of this dataset.

with the ground truth depth range.

On MVSEC, the learnable constant also improves several metrics, but the effect is less uniform. For FullyConv, the improvement is consistent, with Abs Rel decreasing from 0.375 to 0.362 and RMSE decreasing from 6.673 to 6.501. For U-Net, Abs Rel and RMSE improve, but some logarithmic metrics remain weaker than the best U-Net architecture variant. Overall, making the inversion constant learnable is beneficial for in-domain performance, especially on DSEC and for FullyConv on MVSEC.

Overall, the in-domain ablation study shows that the learned representation is sensitive to both output design and model capacity. Three-channel outputs remain preferable, the sigmoid activation is stable but not critical, reduced U-Net capacity can help reduce overfitting on MVSEC, and a learnable inversion constant improves the alignment between DAv2 predictions and the target depth range. These findings suggest that the representation learner is not simply reconstructing an image-like input, but learns a dataset-dependent adaptation to the frozen DAv2 backbone.

4.6. Cross-Dataset Ablation Studies

In this section, we analyze whether the design choices from the in-domain ablation study also improve cross-dataset generalization. We train on one dataset and evaluate on the other dataset without additional fine-tuning. The results are shown in Table 4. Overall, the results show that design choices that improve in-domain performance do not necessarily improve transfer performance. This suggests that the representation learner can still adapt to dataset-specific event statistics, even when the DAv2 backbone remains frozen.

4.6.1. Grayscale Representation

In this experiment, we evaluate whether using a grayscale representation improves generalization across datasets. Similar to the in-domain setting, the representation learner

outputs a single channel, which is duplicated to three channels before being passed to DAv2.

For the DSEC to MVSEC setting, the grayscale representation reduces performance for both architectures. This indicates that removing the channel dimension makes the learned representation less transferable in this direction. For MVSEC to DSEC, the effect is smaller, especially for FullyConv, but the grayscale representation still does not provide a consistent improvement across all metrics. Overall, these results show that the three-channel representation remains preferable for cross-dataset transfer.

4.6.2. Output Activation

In this experiment, we evaluate the effect of replacing the sigmoid output activation of the U-Net with a ReLU activation.

The effect of the ReLU activation depends strongly on the transfer direction. For DSEC to MVSEC, ReLU reduces performance compared to the sigmoid baseline. In contrast, for MVSEC to DSEC, ReLU improves over the baseline across most metrics. This suggests that the best output activation depends on the source dataset. While the bounded sigmoid output is more stable in-domain and for DSEC to MVSEC transfer, the less constrained ReLU output appears to help when training on MVSEC and evaluating on DSEC.

4.6.3. U-Net Architecture Variants

In this experiment, we analyze the effect of reducing the width and depth of the U-Net representation learner.

For DSEC to MVSEC, reducing the number of base channels improves several metrics, although the default U-Net remains competitive. The shallower one-encoder/one-decoder variant does not show the same benefit in this direction. For MVSEC to DSEC, both reduced-capacity variants improve over the default U-Net, with the shallow U-Net achieving the best performance among the U-Net variants. This suggests that lower-capacity models can reduce overfitting to the source dataset, especially when the source dataset is MVSEC. Since MVSEC has lower resolution and less diverse training data, the default U-Net may learn source-specific representations that transfer poorly to DSEC.

4.6.4. Learnable Inversion Constant

In this experiment, we evaluate whether the learnable inversion constant improves cross-dataset generalization.

For DSEC to MVSEC, the effect is mixed. The learnable constant improves some metrics, but reduces performance on others compared to the corresponding baselines. This indicates that the inversion parameter can adapt well to the source dataset, but does not necessarily transfer to the target depth distribution. For MVSEC to DSEC, the effect is more positive, especially for FullyConv, where the learnable constant improves all metrics compared to the baseline.

Overall, the cross-dataset ablations show that there is no single design choice that consistently improves generalization in both transfer directions. Three-channel representations remain preferable, reduced U-Net capacity can help when training on MVSEC, and the learnable inversion constant is useful in some settings but not universally. This indicates that learning the event representation improves in-domain performance, but cross-dataset generalization remains sensitive to the source dataset, training diversity, and the design of the representation learner.

4.7. Stability Analysis

To assess the stability of the main DSEC results, we repeat selected experiments with five independent training runs using five different fixed seeds: 1, 2, 3, 4, and 5. These results should be interpreted as a stability analysis of the main in-domain DSEC findings, rather than as a complete statistical significance test across all datasets and compared methods.

For each repeated experiment, we report the mean and standard deviation across the five runs. This allows us to evaluate whether the observed DSEC performance differences are robust to variation caused by training initialization and stochastic training effects. In particular, we focus on the default U-Net DAv2 and FullyConv DAv2 settings, since these correspond to the main proposed models. The results are shown in Table 5.

4.8. Runtime and Memory Requirements

The runtime and memory requirements are presented in Table 6. The reported results are obtained by averaging the forward and backward passes over a single sequence from the DSEC dataset.

For the ViT-S backbone, which is used in our main experiments, the step time is comparable across models. The U-Net is approximately 2% faster than DAE, while the FullyConv model is 96% slower, despite having fewer trainable parameters than the U-Net. This shows that the number of trainable parameters alone does not determine the computational cost. A likely explanation is that the U-Net performs spatial downsampling, reducing the computational cost, whereas the FullyConv model maintains the full spatial resolution throughout the network.

Another factor is that, although only the representation learner is optimized and DAv2 is kept frozen, gradients still need to propagate through the frozen VFM during backpropagation. This results in a relatively high computational cost for both methods, and helps explain why the U-Net remains close to DAE in runtime while the FullyConv variant is much slower.

The difference in computational cost becomes more important for larger backbones. For the ViT-B backbone, U-Net introduces a 9% increase in step time compared to DAE, while for ViT-L this increases to 18%. This indicates

Table 5. Repeated-run stability analysis on DSEC over five independent training runs using fixed seeds 1, 2, 3, 4, and 5. We report the mean and standard deviation for the main proposed models.

Model	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	SI log ↓	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
U-Net DAv2	0.150 ± 0.002	0.050 ± 0.001	7.200 ± 0.070	0.197 ± 0.002	0.042 ± 0.001	0.794 ± 0.004	0.957 ± 0.001	0.989 ± 0.000
FullyConv DAv2	0.148 ± 0.002	0.048 ± 0.001	7.129 ± 0.059	0.195 ± 0.002	0.042 ± 0.001	0.799 ± 0.004	0.957 ± 0.002	0.989 ± 0.001

Table 6. Runtime, memory usage, and trainable parameter count for different models and DAv2 backbone variants. Step time denotes the forward and backward pass, GPU indicates peak memory usage, and Params refers to the number of trainable parameters. U-Net provides the best efficiency trade-off among the proposed adapters, while FullyConv uses very few trainable parameters but is slower due to full-resolution processing.

Backbone	Model	Step (ms)	GPU (MB)	Trainable Params
ViT-S	DAE	32.6	1114	24.8M
	FullyConv	64.0	2293	0.08M
	U-Net	31.8	1457	0.12M
ViT-B	DAE	58.9	3240	97.5M
	FullyConv	85.1	4604	0.08M
	U-Net	53.4	4063	0.12M
ViT-L	DAE	161.1	10160	335M
	FullyConv	163.9	12042	0.08M
	U-Net	132.6	12456	0.12M

that the efficiency of the representation learner becomes increasingly important as the backbone size grows. Overall, U-Net provides the best efficiency trade-off among the proposed adapters, while the FullyConv model uses fewer trainable parameters but is slower due to full-resolution processing.

5. Conclusion

In this work, we explored event-based monocular depth estimation using a frozen DAv2 backbone and a learned event representation. We evaluated two representation learners, a U-Net and a FullyConv model, to study whether event data can be adapted to an RGB-based VFM without fine-tuning the backbone.

In the in-domain setting, both models perform strongly on DSEC, outperforming all baselines, including RGB input. This shows that learning the event representation can be an effective way to adapt event data to the input space expected by DAv2. On MVSEC, the results are more mixed. The FullyConv model is slightly better than DAE on several metrics, while U-Net remains competitive but does not outperform the strongest baselines. Overall, the FullyConv variant achieves the best accuracy, while the gain on MVSEC remains limited.

For cross-dataset generalization, the results do not show a consistent benefit from keeping the DAv2 backbone frozen. The FullyConv model transfers reasonably well from DSEC to MVSEC and performs competitively with Depth AnyEvent in this setting. However, this trend does not hold in the reverse direction. When training on MVSEC and evaluating on DSEC, both representation learners underperform compared to DAE. This indicates that the learned representation can still overfit to the source dataset, even when the DAv2 backbone remains fixed.

A possible direction for improvement is to pretrain the representation learner on a larger and more diverse dataset. Notably, DAE benefits from pretraining on the synthetic EventScape dataset [10], which likely improves its robustness under domain shift.

In terms of efficiency, U-Net provides the best trade-off. It is faster than the FullyConv model and remains close to DAE in runtime, while using far fewer trainable parameters. The FullyConv model achieves slightly better accuracy, but is slower because it preserves the full spatial resolution throughout the network. This highlights a trade-off between accuracy and computational cost.

Overall, our method is effective for in-domain event-based depth estimation, especially on DSEC. However, learning the representation alone does not provide a clear solution to cross-dataset domain shift. Future work could explore synthetic pretraining, stronger regularization, or jointly learning representations across multiple datasets to improve the robustness of the learned event representation.

References

[1] Luca Bartolomei, Enrico Mannocci, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. Depth AnyEvent: A Cross-Modal Distillation Paradigm for Event-Based Monocular Depth Estimation, 2025. arXiv:2509.15224 [cs]. 1, 2, 3, 4, 5, 6, 7

[2] Aleksei Bochkovskii, AmaãĢl Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R Richter, and Vladlen Koltun. Depth pro: Sharp monocular metric depth in less than a second. *arXiv preprint arXiv:2410.02073*, 2024. 2

[3] Vincent Brebion, Julien Moreau, and Franck Davoine. DELTA: Dense Depth from Events and LiDAR using Transformer's Attention. 2

[4] Vincent Brebion, Julien Moreau, and Franck Davoine. Learning to Estimate Two Dense Depths from LiDAR and Event Data. In *Image Analysis*, pages 517–533. Springer Na-

ture Switzerland, Cham, 2023. Series Title: Lecture Notes in Computer Science. 2

[5] Qifeng Chen, Jia Xu, and Vladlen Koltun. Fast image processing with fully-convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2497–2506, 2017. 3, 4

[6] Anusha Devulapally, Md Fahim Faysal Khan, Siddharth Advani, and Vijaykrishnan Narayanan. Multi-Modal Fusion of Event and RGB for Monocular Depth Estimation Using a Unified Transformer-based Architecture. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2081–2089, Seattle, WA, USA, 2024. IEEE. 2

[7] Xingshuai Dong, Matthew A. Garratt, Sreenatha G. Anavatti, and Hussein A. Abbass. Towards real-time monocular depth estimation for robotics: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):16940–16961, 2022. 1

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[9] Ashkan Ganj, Yiqin Zhao, Hang Su, and Tian Guo. Mobile ar depth estimation: Challenges & prospects. In *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, page 21–26, New York, NY, USA, 2024. Association for Computing Machinery. 1

[10] Daniel Gehrig, Michelle Rüegg, Mathias Gehrig, Javier Hidalgo-Carrió, and Davide Scaramuzza. Combining events and frames using recurrent asynchronous multimodal networks for monocular depth prediction. *IEEE Robotics and Automation Letters*, 6(2):2822–2829, 2021. 7, 12

[11] Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. DSEC: A Stereo Event Camera Dataset for Driving Scenarios, 2021. arXiv:2103.06011 [cs]. 5

[12] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017. 2

[13] Javier Hidalgo-Carrió, Daniel Gehrig, and Davide Scaramuzza. Learning Monocular Dense Depth from Events, 2020. arXiv:2010.08350 [cs]. 2

[14] Ze Huang, Li Sun, Cheng Zhao, Song Li, and Songzhi Su. EventPoint: Self-Supervised Interest Point Detection and Description for Event-based Camera. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5385–5394, Waikoloa, HI, USA, 2023. IEEE. 3

[15] Chenxu Jiang, Mingyuan Lin, Chi Zhang, Zhenghai Wang, and Lei Yu. Learning Monocular Depth from Focus with Event Focal Stack, 2024. arXiv:2405.06944 [cs]. 2

[16] Jie Long Lee and Gim Hee Lee. Distil-E2D: Distilling Image-to-Depth Priors for Event-Based Monocular Depth Estimation. 2

[17] Haotong Lin, Sili Chen, Junhao Liew, Donny Y Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025. 2

[18] Haotian Liu, Sanqing Qu, Fan Lu, Zongtao Bu, Florian Roehrbein, Alois Knoll, and Guang Chen. PCDepth: Pattern-based Complementary Learning for Monocular Depth Estimation by Best of Both Worlds, 2024. arXiv:2402.18925 [cs]. 2

[19] Xu Liu, Jianing Li, Xiaopeng Fan, and Yonghong Tian. Event-based Monocular Dense Depth Estimation with Recurrent Transformers, 2022. arXiv:2212.02791 [cs]. 2

[20] Haitao Meng, Chonghao Zhong, Sheng Tang, Lian JunJia, Wenwei Lin, Zhenshan Bing, Yi Chang, Gang Chen, and Alois Knoll. Learning Monocular Depth from Events via Egomotion Compensation, 2024. arXiv:2412.19067 [cs]. 2

[21] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning Robust Visual Features without Supervision, 2024. arXiv:2304.07193 [cs]. 2

[22] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1623–1637, 2020. 2, 4

[23] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision Transformers for Dense Prediction, 2021. arXiv:2103.13413 [cs]. 2

[24] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE transactions on pattern analysis and machine intelligence*, 43(6):1964–1980, 2019. 5, 6

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3

[26] Florian Sauerbeck, Dan Halperin, Lukas Connert, and Johannes Betz. Camradepth: Semantic guided depth estimation using monocular camera and sparse radar for automotive perception. *IEEE Sensors Journal*, 23(22):28442–28453, 2023. 1

[27] Wenming Weng, Yueyi Zhang, and Zhiwei Xiong. Event-based video reconstruction using transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2543–2552, 2021. 5, 6

[28] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data, 2024. arXiv:2401.10891 [cs]. 2

[29] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth Anything V2, 2024. arXiv:2406.09414 [cs]. 2, 3

[30] Xin Zhang, Liangxiu Han, Tam Sobeih, Lianghao Han, and Darren Dancey. A Novel Spike Transformer Network for Depth Estimation from Event Cameras via Cross-modality Knowledge Distillation, 2025. arXiv:2404.17335 [cs]. 2

[31] Haokun Zheng, Sidhant Rajadnya, and Avideh Zakhor. Monocular depth estimation for drone obstacle avoidance in indoor environments. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10027–10034. IEEE, 2024. 1

[32] Alex Zihao Zhu, Dinesh Thakur, Tolga Ozaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The Multi Vehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018. arXiv:1801.10202 [cs]. 5

[33] Jinjing Zhu, Tianbo Pan, Zidong Cao, Yexin Liu, James T Kwok, and Hui Xiong. Depth Any Event Stream: Enhancing Event-based Monocular Depth Estimation via Dense-to-Sparse Distillation. 2, 3

References

[1] Jay Alammar. *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/. Accessed: 2026-05-25. 2018.

[2] Luca Bartolomei et al. “Depth AnyEvent: A Cross-Modal Distillation Paradigm for Event-Based Monocular Depth Estimation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2025, pp. 19669–19678.

[3] Mathilde Caron et al. “Emerging properties in self-supervised vision transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660.

[4] Xingshuai Dong et al. “Towards Real-Time Monocular Depth Estimation for Robotics: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.10 (2022), pp. 16940–16961. DOI: 10.1109/TITS.2022.3160741.

[5] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[6] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[7] David Eigen, Christian Puhrsch, and Rob Fergus. “Depth map prediction from a single image using a multi-scale deep network”. In: *Advances in neural information processing systems* 27 (2014).

[8] Sanja Fidler. *Depth from Stereo*. https://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12.pdf. CSC420: Introduction to Image Understanding, University of Toronto. Accessed: 2026-04-22. 2015.

[9] Guillermo Gallego et al. “Event-based vision: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[10] Ashkan Ganj et al. “Mobile AR depth estimation: Challenges & prospects”. In: *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*. 2024, pp. 21–26.

[11] Daniel Gehrig and Davide Scaramuzza. “Low-latency automotive vision with event cameras”. In: *Nature* 629.8014 (2024), pp. 1034–1040.

[12] Daniel Gehrig et al. “Combining events and frames using recurrent asynchronous multimodal networks for monocular depth prediction”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2822–2829.

[13] Mathias Gehrig et al. “Dsec: A stereo event camera dataset for driving scenarios”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4947–4954.

[14] Clément Godard et al. “Digging into self-supervised monocular depth estimation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3828–3838.

[15] Julian Habekost et al. “From Synthetic to One-Shot Regression of Camera-Agnostic Human Performances”. In: *Pattern Recognition and Artificial Intelligence: Third International Conference, ICPRAI 2022, Paris, France, June 1–3, 2022, Proceedings, Part I*. Paris, France: Springer-Verlag, 2022, pp. 514–525. ISBN: 978-3-031-09036-3. DOI: 10.1007/978-3-031-09037-0_42. URL: https://doi.org/10.1007/978-3-031-09037-0_42.

[16] Jiajun He et al. “Bearing Fault Diagnosis via Improved One-Dimensional Multi-Scale Dilated CNN”. In: *Sensors* 21.21 (2021). ISSN: 1424-8220. DOI: 10.3390/s21217319. URL: https://www.mdpi.com/1424-8220/21/21/7319.

[17] Javier Hidalgo-Carrió, Daniel Gehrig, and Davide Scaramuzza. “Learning monocular dense depth from events”. In: *2020 International Conference on 3D Vision (3DV)*. IEEE. 2020, pp. 534–542.

[18] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[19] Ze Huang et al. “Eventpoint: Self-supervised interest point detection and description for event-based camera”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 5396–5405.

[20] Alexander Kirillov et al. “Segment anything”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2023, pp. 4015–4026.

[21] Andrej Krenker, Janez Bešter, and Andrej Kos. “Introduction to the artificial neural networks”. In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech* (2011), pp. 1–18.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[23] Xavier Lagorce et al. “HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.7 (2017), pp. 1346–1359. DOI: 10.1109/TPAMI.2016.2574707.

[24] Iro Laina et al. “Deeper depth prediction with fully convolutional residual networks”. In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 239–248.

[25] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[26] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. “A 128$\times$ 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor”. In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576. DOI: 10.1109/JSSC.2007.914337.

[27] Luxonis. *Configuring Stereo Depth*. Accessed: 2026-04-21. 2024. URL: https://docs.luxonis.com/hardware/platform/depth/configuring-stereo-depth.

[28] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. en. arXiv:2304.07193 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2304.07193. URL: http://arxiv.org/abs/2304.07193 (visited on 11/26/2025).

[29] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. “Vision transformers for dense prediction”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 12179–12188.

[30] René Ranftl et al. “Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.3 (2020), pp. 1623–1637.

[31] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. “Esim: an open event camera simulator”. In: *Conference on robot learning*. PMLR. 2018, pp. 969–982.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[33] Florian Sauerbeck et al. “CamRaDepth: Semantic Guided Depth Estimation Using Monocular Camera and Sparse Radar for Automotive Perception”. In: *IEEE Sensors Journal* 23.22 (2023), pp. 28442–28453. DOI: 10.1109/JSEN.2023.3321886.

[34] Nathaniel Simon and Anirudha Majumdar. “Mononav: Mav navigation via monocular depth estimation and reconstruction”. In: *International Symposium on Experimental Robotics*. Springer. 2023, pp. 415–426.

[35] Tomasz Szandała. “Review and comparison of commonly used activation functions for deep neural networks”. In: *Bio-inspired neurocomputing*. Springer, 2020, pp. 203–224.

[36] user231225. *How would i draw a Graph like that with LaTeX?* https://tex.stackexchange.com/questions/576142/how-would-i-draw-a-graph-like-that-with-latex. TeX StackExchange answer. Retrieved 2026-05-25. Licensed under CC BY-SA 4.0. Dec. 2020.

[37] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[38] Athanasios Voulodimos et al. “Deep learning for computer vision: A brief review”. In: *Computational intelligence and neuroscience* 2018.1 (2018), p. 7068349.

[39] Lin Wang et al. “Event-Based High Dynamic Range Image and Very High Frame Rate Video Generation Using Conditional Generative Adversarial Networks”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10073–10082. DOI: 10.1109/CVPR.2019.01032.

[40] M. Williams. *Kernel Convolution*. Accessed: 2026-05-08. 2024. URL: https://milliams.com/courses/neural_networks/Kernel%20convolution.html.

[41] Lihe Yang et al. “Depth anything v2”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 21875–21911.

[42] Yao Yao et al. “Mvsnet: Depth inference for unstructured multi-view stereo”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 767–783.

[43] Haokun Zheng, Sidhant Rajadnya, and Avideh Zakhor. “Monocular Depth Estimation for Drone Obstacle Avoidance in Indoor Environments”. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 10027–10034. DOI: 10.1109/IROS58592.2024.10802577.

[44] Alex Zihao Zhu et al. “The multivehicle stereo event camera dataset: An event camera dataset for 3D perception”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2032–2039.