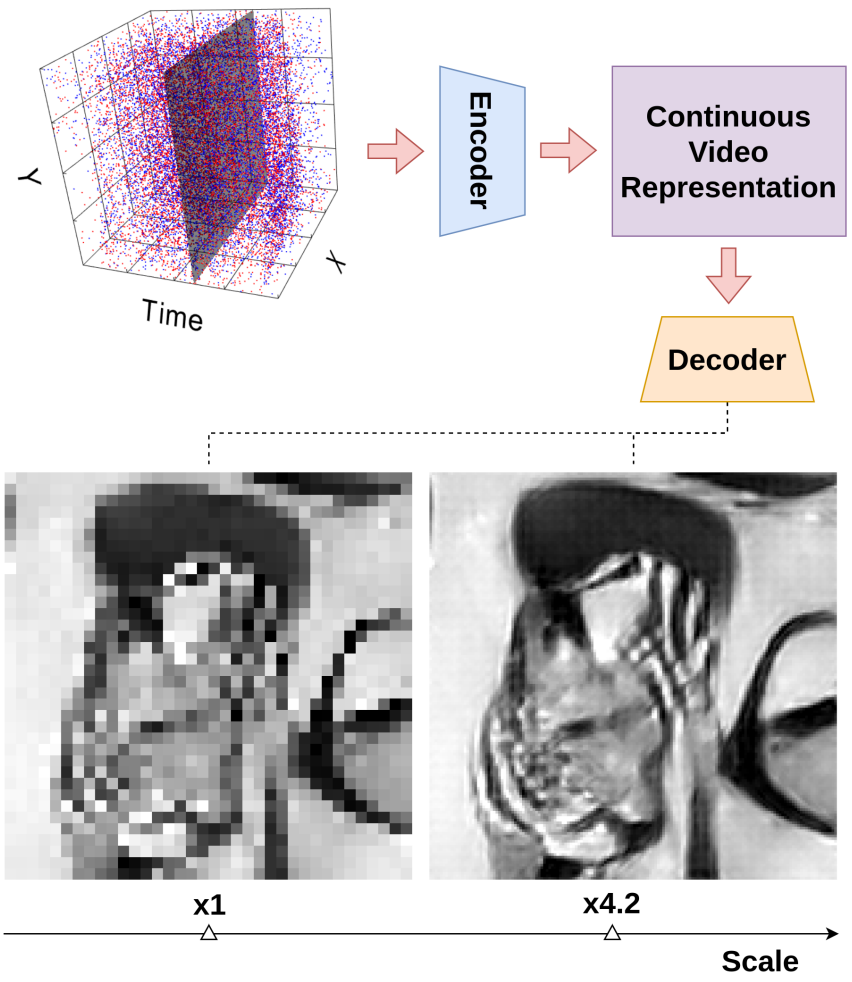
CHALMERS
UNIVERSITY OF TECHNOLOGY



Learning Continuous Video Representation from Event Cameras

A Local Implicit Function approach for video reconstruction and spatiotemporal superresolution

Master's thesis in Complex Adaptive Systems

DAVID TONDERSKI

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Learning Continuous Video Representation from Event Cameras

A Local Implicit Function approach for video reconstruction and spatiotemporal superresolution

David Tonderski

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2024

Learning Continuous Video Representation from Event Cameras
A Local Implicit Function approach for video reconstruction and spatiotemporal superresolution
David Tonderski

Supervisor: Dr. Valery Vishnevskiy, ETH Zurich
Examiner: Prof. Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2024
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Overview of the proposed method. Event data is encoded into a continuous video representation, which can then be queried to generate intensity frames at any continuous timestamp and spatial resolution.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Learning Continuous Video Representation from Event Cameras
A Local Implicit Function approach for video reconstruction and spatiotemporal superresolution
David Tonderski
Department of Physics
Chalmers University of Technology

Abstract

Event cameras are biologically inspired sensors that operate differently from conventional cameras. Rather than measuring pixel intensities at fixed intervals, event cameras detect per-pixel intensity changes, offering high dynamic range, low latency, high temporal resolution, minimal motion blur, and low power consumption. However, traditional computer vision algorithms cannot be applied to event data due to the radically different operating paradigm. One approach to bridge this gap is to reconstruct conventional images from event data. While this approach retains the high dynamic range and minimal motion blur, it does not fully capture the high temporal resolution of event cameras.

In this thesis, we utilize Local Implicit Functions for spatiotemporal video reconstruction, aiming to preserve the high temporal resolution of event data as well as allow for the generation of videos with an arbitrary spatial resolution. We show that our method reaches reconstruction quality similar to comparable state-of-the-art approaches, and significantly outperforms simple baselines for spatial upscaling up to 3x. Our analysis also suggests that our representation retains the high temporal resolution of event data. Additionally, our approach offers per-pixel uncertainty estimations, which have the potential to enhance the performance of downstream computer vision applications.

Keywords: event cameras, reconstruction, superresolution, uncertainty quantification.

Acknowledgements

I would like to thank my supervisor Valery Vishnevskiy for invaluable advice and great discussions throughout the project. Also, thank you to Sony Stuttgart Laboratory 1 for providing crucial resources.

David Tonderski, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

APS	Active-Pixel Sensor
CNN	Convolutional Neural Network
ConvGRU	Convolutional Gated Recurrent Unit
ConvLSTM	Convolutional Long Short-Term Memory
ELU	Exponential Linear Unit
EPF	Event per Pixel per Frame
EVS	Event-based Vision Sensor
LL	Log-Likelihood
LPIPS	Learned Perceptual Image Patch Similarity
LPIPS_CORR	LPIPS and Standard Deviation Correlation Index
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MVE	Mean Variance Estimation
PICP	Prediction Interval Coverage Probability
RCNN	Recurrent Convolutional Neural Network
ReLU	Rectified Linear Unit
SSIM	Structured Similarity Index Measure

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i	Index for sequence
τ	Index for event frame
b	Index for event bin

Sets

ψ	Parameters of decoder
ϕ	Parameters of encoder
ϵ	Set of events
S	Subset of events

Parameters

M	Number of images passed into simulator
N_{max}	Maximum number of transformation control points
B	Number of temporal bins
λ_{MVE}	Weight of MVE loss
γ	Learning rate
N_{iters}	Number of training iterations

Variables

x, y	Pixel coordinates
t	Timestamp
p	Event polarity
L	Log intensity
C	Contrast threshold
h	Neuron state
μ	True mean
σ	True standard deviation
$\hat{\mu}$	Predicted mean
$\hat{\sigma}$	Predicted standard deviation
z	Quantile of probability distribution
ρ	Spatial resolution
A	Affine matrix
N_j	Number of control points for transformation j
α	Rotation transformation values
s_x, s_y	Scaling transformation values
d_x, d_y	Displacement transformation values
C_s, C_α	Scaling and rotation normalization values
p	Control point value
L_{min}, L_{max}	Illuminance range
k	Image exponentiation multiplier
E	Event tensor
z	Latent code
v	Coordinate of latent code
s	Predicted signal
$\mathbf{x}$	3D spatiotemporal coordinate
$M^{(i)}$	Feature map
ν_ϕ	Encoder
C_ψ	Decoder
$\mathcal{L}$	Loss function

Contents

List of Figures

List of Tables

1

Introduction

Event cameras, also called Event-based Vision Sensors (EVS), are biologically inspired sensors with a radically different operating paradigm compared to conventional Active-Pixel Sensor (APS) cameras. Instead of measuring pixel intensities at a fixed rate, they measure per-pixel intensity *changes* independently and asynchronously. Their advantages include a high dynamic range (140 dB vs 60 dB of APS cameras), low latency and high temporal resolution (on the order of microseconds), low motion blur, and low power consumption (~ 10 mW) [1]. However, the unique data acquisition method of event cameras also means that traditional Computer Vision techniques cannot be directly applied to them, and novel methods must be developed. Furthermore, event cameras usually have a limited spatial resolution. While high resolutions cameras exist, Gehrig and Scaramuzza showed that they suffer from increased noise, which can cause lower performance on many vision tasks [2].

In the seminal paper E2VID [3], Rebecq et al present a method for bridging the gap between traditional computer vision and event cameras by reconstructing intensity frames from event data using Recurrent Convolutional Neural Networks (RCNN). They show that their method is able to capture the advantages of event cameras by synthesizing videos of high-speed phenomena as well as providing high dynamic range reconstructions in challenging light conditions. They also show that traditional Computer Vision techniques can be directly applied to their reconstructions, outperforming methods specialized for event data.

Because methods like E2VID output reconstructions at a fixed framerate, they do not retain the high temporal resolution of event cameras[1]. Furthermore, such methods can cause variability in the performance of downstream tasks which cannot be accounted for by the tasks themselves. For example, if the reconstruction algorithms fails to reconstruct a face, then a face detection algorithm applied to the reconstruction would have a high confidence that a face is not present in the data.

In this work, we propose a method to reconstruct videos of arbitrary spatiotemporal resolution from event data. To achieve this, we draw inspiration from the Chen et al. [4], who propose a method to learn a continuous image representation. At

[1] Although very high framerates can be achieved with E2VID, this is done by re-running the method multiple times.

first, a Convolutional Neural Network encodes the input image into a set of spatially distributed latent codes. A Multi-Layer Perceptron (MLP) then predicts the pixel intensity at any continuous image coordinate based on these latent codes, allowing image generation at arbitrary resolutions. Similarly to their follow-up method VideoINR [5], we extend this concept to the temporal dimension. We use a RCNN to encode event data into a spatiotemporally distributed latent codes. Then, we use an MLP to predict image intensities and their variances at continuous spatiotemporal coordinates, enabling the reconstruction of videos at any spatiotemporal resolution. Furthermore, we quantify the uncertainty inherent to our reconstructions by using a Mean Variance Estimation (MVE) network [6]. The MVE network is designed to predict the mean and variance of the target distribution at each pixel, assuming the errors follow a normal distribution. In practice, this means that we get a pixel-by-pixel measure of the network uncertainty. We show an overview of our method in figure 1.1.

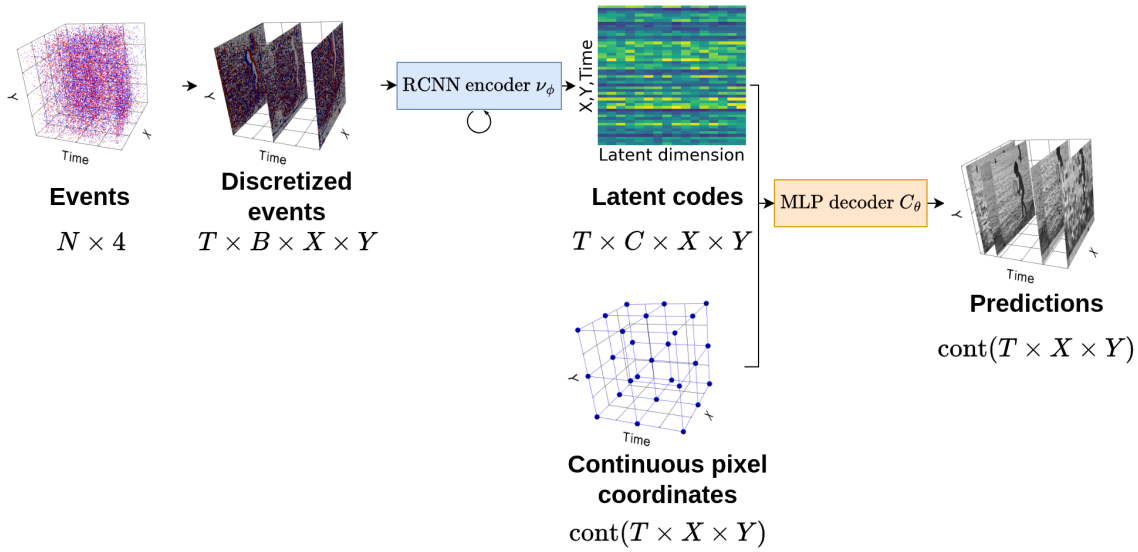


Figure 1.1: Overview of our method. Events are discretized and encoded by a RCNN into latent codes. These latent codes are then used by a MLP decoder to predict intensities and uncertainty estimates at spatiotemporally continuous pixel coordinates.

1.1 Limitations

Although intensity reconstruction from event data is frequently considered in the context of applying computer vision algorithms to the reconstructions, we do not investigate this. We also do not investigate how the predicted uncertainties could be used to improve the performance of such algorithms. When evaluating spatiotemporal superresolution, we do not provide quantitative results on real data, as we did not find a suitable dataset with high FPS and high resolution intensity images. We also do not compare the superresolution performance of our method to other methods.

2

Background

This section introduces event cameras and deep learning, and provides an overview of related literature.

2.1 Event cameras

In contrast to standard cameras, which output the brightness of all pixels at a fixed rate, event cameras respond to brightness changes asynchronously and independently for each pixel [1]. They output a stream of *events*, with each event consisting of a discrete pixel coordinate (x, y), a continuous timestamp t, and a binary polarity p. Each event represents a change of brightness of a pixel at a specific time, and the polarity determines whether the brightness increased or decreased.

In their survey, Gallego et al. identify four key advantages of event cameras [1]:

1. Temporal resolution - events are read with a clock with a frequency on the order of 1 MHz [1], so events are captured with a temporal resolution on the order of microseconds. This lets event cameras capture very fast motion without motion blur.

2. Low latency - since each pixel transmits events asynchronously and independently, event cameras have latencies as low as 15 µs [7], making them especially suitable for real-time systems.

3. Low power - event cameras do not process redundant brightness data, and they do not consume power unless the brightness changes. This leads to a drastically lower power consumption compared to traditional cameras (on the order of milliwatts instead of watts [2].

4. High dynamic range - because the pixel photoreceptors operate in a logarithmic scale, event cameras have a drastically higher dynamic range than standard cameras (120 dB compared to 60 dB).

However, because the data they generate is fundamentally different than regular cameras, traditional computer vision algorithms are not directly applicable to event

cameras. Furthermore, event cameras are inherently noisier than traditional cameras [1]. Therefore, novel methods are required to extract information from event data.

Due to these trade-offs, event cameras are typically used in situations requiring low power consumption, low latency, and robustness to varying lighting conditions. Examples include wearable electronics [1], robotics [8]–[10], tactile sensing [11], [12], and high-speed control [13], [14].

Each pixel in an event camera responds to changes in log intensity $L = \log(I)$. In an ideal, noise-free scenario, an event $e_k = (x_k, y_k, t_k, p_k)$ is emitted at pixel (x_k, y_k) and time t_k when the change in log intensity

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

surpasses a *contrast threshold* C, i.e.

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

where $C > 0$, and $p_k \in \{-1, +1\}$ indicates the direction of the brightness change. The contrast threshold C can be controlled by the user. In practice, the effective contrast thresholds vary from pixel to pixel due to circuit noise and sensor non-idealities, and follow a distribution with an average value of C [1], [15].

2.2 Deep Learning

Deep Learning is a computational methodology that allows algorithms to learn from observational data. It employs structures known as Artificial Neural Networks (ANNs), which mimic the interconnected neurons in the brain. The fundamental unit of these networks is the neuron, which processes incoming data and passes it on to subsequent neurons based on the strength of connections, determined by the weights of the ANN. These neurons are organized into layers, with each layer's neurons receiving inputs only from the preceding layer. This organization enables each layer to transform its input data into increasingly abstract representations; for instance, initial layers in a computer vision model might detect simple features such as edges or lines, while deeper layers might recognize complex objects. Crucially, during the training process, the network learns to extract these features autonomously. The weights are initially set randomly and are subsequently updated to minimize a specific objective function, typically through gradient descent.

Deep learning has made a significant impact across various domains, ranging from healthcare to autonomous driving. In healthcare, it can enhance medical diagnostics by analyzing medical images, such as detecting tumors from MRIs. In autonomous driving, it processes real-time sensor data, helping vehicles navigate safely by recognizing objects and predicting other road users' behaviors. In language processing, it enables applications like ChatGPT to understand and generate human language.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are specialized in processing data with grid-like topologies, such as images. They differ significantly from traditional fully connected neural network architectures. Unlike fully connected layers, where every neuron is connected to every neuron in the previous layer, convolutional layers are structured such that each neuron is only connected to a small region of the previous layer. This design is inspired by the biological processes in the human visual cortex. By focusing on local regions, convolutional layers can more effectively capture spatial hierarchies. Additionally, the weights are shared across all neurons in each layer, improving both learning and computational efficiency. This shared weight approach also provides translation invariance, enabling the network to recognize features regardless of their position in the image. These characteristics make CNNs particularly powerful for tasks like image and video analysis.

2.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural networks designed to handle sequential data by maintaining a form of memory. Unlike traditional neural networks, RNNs have connections that form directed cycles, allowing information to persist across time steps. This architecture is particularly well-suited for tasks such as language modeling, time series prediction, and speech recognition.

In an RNN, the state of neurons is given by

$$h_{l,t} = f(h_{l,t-1}, h_{l-1,t}), \tag{2.3}$$

where $h_{l,t}$ denotes the state of neurons at layer l and timestamp t. The function f depends on the network architecture. In this thesis, we use Gated Recurrent Units (GRUs).

GRUs are a type of RNN architecture designed to address some of the limitations of standard RNNs, particularly the vanishing gradient problem, which makes it difficult for the network to learn long-term dependencies in sequential data. GRUs introduce gating mechanisms that control the flow of information from previous time steps, enabling the network to maintain memory more effectively.

GRUs use two gates: the update gate and the reset gate. The update gate determines how much of the past information needs to be passed along to the future, while the reset gate decides how much of the past information to forget. These gates allow the GRU to keep relevant information from previous time steps and discard irrelevant information, making it easier to capture long-term dependencies.

2.2.3 Implicit neural representations

Implicit Neural Representations (INRs), also known as coordinate-based representations, present a novel method for parameterizing continuous signals [16]. Standard signal representations of signals are usually discrete - for example pixel grids in images, amplitude samples in audio, or voxel grids and meshes in 3D models. INRs, by

contrast, parametrize these signals as continuous functions from the signal domain (for example, a 2D pixel coordinate) to its value (e.g. RGB color). Since these continuous functions are not analytically tractable, they are instead approximated by a neural network.

INRs are having an impact across various domains due to their unique advantages and applications. One of their most notable benefits is memory efficiency, allowing them to represent complex signals without the large memory footprint required by traditional methods like pixel or voxel grids. Additionally, INRs are not constrained by fixed resolutions; they can generate data at any desired resolution. A key application of INRs is in the creation of Neural Radiance Fields (NeRFs), which are used for reconstructing and rendering intricate 3D scenes from sparse and irregularly sampled images. NeRFs employ a neural network to map spatial and directional inputs to color and density, enabling photorealistic renderings from novel viewpoints.

2.2.4 Mean-Variance Estimation Networks

A popular way of quantifying the uncertainty of neural networks is through Prediction Intervals (PI), as they are one of the most understandable uncertainty prediction mechanisms [17]. Nix and Weighed proposed [6] to construct these intervals through Mean-Variance Estimation networks. They build on the assumption that

$$y|x \sim \mathcal{N}(\mu, \sigma^2), \tag{2.4}$$

where μ and σ^2 are the mean and variance of the conditional distribution of target variable y given an input x. In a traditional regression task, we would then train a neural network f_θ to output estimates $\hat{\mu}(x)$ of the true mean $\mu(x)$. In MVE networks, the network additionally outputs an estimate $\hat{\sigma}$ estimating the true variance σ of that distribution. Then, the $(1-\alpha)\%$ prediction interval can be formed as

$$\hat{y}(x) \pm z_{1-\frac{\alpha}{2}}\hat{\sigma}(x), \tag{2.5}$$

where $z_{1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$ quantile of the $\mathcal{N}(0,1)$ probability distribution. These networks are usually trained through Maximum Likelihood Estimation [17]. We begin by expressing the log likelihood of the targets given the inputs and a network f_θ as

$$\begin{aligned} \ln p(\mu_i|x_i, f_{\theta,\mu}, f_{\theta,\sigma}) &= \ln\left[\frac{1}{\sqrt{2\pi\hat{\sigma}^2(x_i)}}\exp\left(\frac{-(\mu_i-\hat{\mu}(x_i))^2}{2\hat{\sigma}^2(x_i)}\right)\right] \\ &= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\hat{\sigma}^2(x_i)) - \frac{(\mu_i-\hat{\mu}(x_i))^2}{2\hat{\sigma}^2(x_i)}. \end{aligned} \tag{2.6}$$

Then, ignoring constants, we can define the loss function to be minimized as:

$$\mathcal{L}_{MVE}(\hat{\mu}, \hat{\sigma}, \mu) = \sum_i \frac{1}{2}\ln(\hat{\sigma}_i^2) + \frac{(\mu_i-\hat{\mu}_i)^2}{2\hat{\sigma}_i^2}, \tag{2.7}$$

which is precisely the loss function used in MVE networks [6].

2.3 Related literature

2.3.1 Image reconstruction from event data

Events can be interpreted as providing information about the derivative of the log intensity. Assuming small Δt_k, we can use equation 2.2 to approximate the derivative as

$$\frac{\partial L}{\partial t}(x_k, y_k, t_k) \approx \frac{p_k C}{t_k}. \tag{2.8}$$

This interpretation has been used to reconstruct intensity images in algorithms such as [18]–[21]. However, these approaches can suffer from loss of detail, visual artifacts, and reliance on hand-crafted priors [3]. In recent years, the state-of-the-art has shifted to reconstructing intensity images using deep learning.

Rebecq et al. [3] first proposed the E2VID model, which uses a RCNN to achieve a significant performance improvement compared to hand-crafted methods. Schneerlink et al. [22] then significantly reduced the network complexity with only a minor drop in prediction quality through the FireNET model. Then, Stoffregen et al. [23] showed that the performance of these methods is limited by the quality of the simulated data used to train them. By increasing the realism of the data, they provided the E2VID+ and FireNET+ models with significantly improved performance compared to the base models. Wang et al. [24] shifted from a convolutional to an attention-based architecture, improving reconstruction quality at the cost of inference time and computational complexity. Finally, Ercan et al. [25] provided a comprehensive framework to evaluate and compare these methods.

2.3.2 Spatiotemporal Video Superresolution

Spatiotemporal Video Superresolution (STSVR) aims to simultaneously increase the spatial and temporal resolution of videos. This is related to our method even though the input is different (events instead of videos), as we aim to provide a video representation of arbitrary spatiotemporal resolution. While researchers have tackled this problem with traditional algorithms [26], [27], they have lately started to employ deep learning-based solutions. In Zooming Slow-Mo [28], Xiang et al. proposed a method for temporal interpolation of the missing frame features, followed by a ConvLSTM for aligning and aggregating information before frame reconstruction. Haris et al. [29] leverage the relationship by space and time by integrating optical flow into their method. Chen et al. [5] leveraged INRs to construct a continuous video representation that can be queried at any spatial and temporal resolution. This is similar to our work, although they leverage optical flow and only consider information within consecutive frames.

3

Methods

3.1 Training data

Our method requires training data in the form of pairs of event sequences and corresponding ground-truth image sequences. While such datasets do exist (e.g. [23], [30]), we also require the ability to generate ground truth images at continuous resolutions and timestamps. Because of this we train the network using synthetically generated data, and later show that it generalizes to real data.

Our approach is inspired by E2VID [3], where they map MS-COCO images to a plane in a 3D space and simulate events by moving a camera randomly in this 3D space using the ESIM event simulator [31]. However, the ESIM simulator does not realistically model sensor noise and non-idealities [31], which hinders generalization to real data. Instead, we use a proprietary, physically accurate event simulator provided by Sony Semiconductor Solutions. It accurately models the sensor circuitry and accounts for non-idealities such as transistor mismatch, background activity, circuit block frequency responses, refractory periods, temperature dependence, and read-out delays [15]. However, this simulator has a different operation principle than the ESIM simulator. Instead of simulating events directly from a 3D scene, it converts sequences of grayscale images into events.

To train our network, we require event sequences $\epsilon^{(i)} = \{e_1^{(i)}, ..., e_K^{(i)}\}$ generated at some resolution $\rho_{min} \in \mathbb{N}^2$ as well as a method to generate corresponding ground truth intensities $\mu^{(i)}(t, x, y)$ at continuous timestamps t and spatial coordinates x, y. To achieve this, we first collect a set of grayscale images $I^{(i)}$ of some resolutions $\rho_{max}^{(i)}$. Specifically, we draw 550 random images (500 for the training set, 50 for the test set) from the MS-COCO dataset [32], convert them to grayscale, and normalize them to $[0, 1]$. Then, for each image, we define temporally continuous affine transforms $f_{affine}^{(i)}(t)$, thus yielding

$$y^{(i)}(t, \rho_{max}^{(i)}) = f_{affine}^{(i)}(t), \tag{3.1}$$

where $y^{(i)}(t, \rho_{max}^{(i)})$ is an image of resolution $\rho_{max}^{(i)} \in \mathbb{R}^2$ containing the ground truth intensities. To get a pixel intensity at a continuous spatiotemporal coordinate (t^*, x^*, y^*), we can simply cubically interpolate the intensities in $y_{max}^{(i)}(t^*, \rho_{max}^{(i)})$ at coordinates (x^*, y^*). This gives us images at any resolution, so why do we need generate events at a lower resolution ρ_{min}? If we didn't, our spatial superresolu-

tion quality would be limited to at best cubic interpolation. Thus, we must choose some maximum scaling factor, and then generate sequences $\epsilon^{(i)}$ at a lower resolution $\rho_{min}^{(i)} = \frac{\rho_{max}^{(i)}}{S_{max}}$:

$$\epsilon^{(i)} \leftarrow \text{Simul}(G^{(i)}(y^{(i)}(t_1, \rho_{min}^{(i)})), ..., G^{(i)}(y^{(i)}(t_M, \rho_{min}^{(i)}))), \tag{3.2}$$

where M denotes the total number of images passed into the simulator, $G^{(i)}$ is a stochastic pre-processing function, and $y^{(i)}(t_m, \rho_{min}^{(i)})$ is generated by cubic downsampling.

3.1.1 Ground truth generation

First, we must define $f_{affine}^{(i)}(t)$ for each sequence i. We do this by constructing temporally continuous affine matrices $A^{(i)}(t)$ as follows:

1. For each sequence i, we define transformation control points for rotation $N_\alpha^{(i)}$, displacement $N_d^{(i)}$, and scale $N_s^{(i)}$. First, we sample the number of control points from a uniform integer distribution

   $$N_j^{(i)} \sim U\{2, N_{max}\},$$

   where j represents each type of transformation.

2. At each control point $k \in \{1, ..., N_j^{(i)}\}$, transformation values are sampled as follows:

   $$\begin{aligned} s_{x,k}^{(i)}, s_{y,k}^{(i)} &\sim U[0, s_{max}], \\ \alpha_k^{(i)} &\sim U[0, \alpha_{max}], \\ d_{x,k}^{(i)}, d_{y,k}^{(i)} &\sim U[-d_{max}, d_{max}], \end{aligned}$$

3. These values are then further scaled and normalized sequence-wise:

   $$\begin{aligned} \hat{s}_{x,k}^{(i)} &= 1 + C_s^{(i)}(s_{x,k}^{(i)} - s_{x,1}^{(i)}), \\ \hat{s}_{y,k}^{(i)} &= 1 + C_s^{(i)}(s_{y,k}^{(i)} - s_{y,1}^{(i)}), \\ \hat{\alpha}_k^{(i)} &= C_\alpha^{(i)}(\alpha_k^{(i)} - \alpha_1^{(i)}), \\ \hat{d}_{x,k}^{(i)} &= d_{x,k}^{(i)} - d_{x,1}^{(i)}, \\ \hat{d}_{y,k}^{(i)} &= d_{y,k}^{(i)} - d_{y,1}^{(i)}, \end{aligned}$$

   where $C_s^{(i)} \sim U[0, 1]$ and $C_\alpha^{(i)} \sim U[0, 1]$. The additional scaling was derived empirically by qualitatively comparing the motion in the generated sequences to the motion in the E2VID dataset. Intuitively, it makes smaller rotation and scaling values more likely, while keeping the possible value range intact. The purpose of the subtractions and additions is to make sure that the first image of the sequence is exactly the original image $I^{(i)}$.

4. To generate the transformation values for a timestamp t, we first associate the control points for interpolation type j with

$$t_{j,k}^{(i)} = \frac{k-1}{N_j^{(i)} - 1}. \tag{3.3}$$

Then, we interpolate the control point values using either linear or cubic interpolation, with a $\frac{1}{2}$ probability of either interpolation type, as we want to learn both smooth and sharp motion changes, see figure 3.1. For linear interpolation of a control value $p_j^{(i)}$ at some time t between control points k^* and k^*+1, the interpolated value $p_j^{(i)}(t)$ is given by:

$$p_j^{(i)}(t) = p_{k^*}^{(i)} + \frac{(t - t_{j,k^*}^{(i)})}{(t_{j,k^*+1}^{(i)} - t_{j,k^*}^{(i)})}(p_{j,k^*+1}^{(i)} - p_{j,k^*}^{(i)}) \tag{3.4}$$

For cubic interpolation, the formulas are more complex and will not be shown here. We use the *interp1d* function from the *scipy* library [33].

5. Next, we generate the affine matrices for each sequence i, timestamp t, and transformation type j. Note that we compute the transformed images using the *affine_grid* and *grid_sample* methods from the *Pytorch* library [34]. These methods expect the affine matrices to define a mapping from the output image to the input image, so the transformation matrices are inverted compared to their standard form:

$$S^{(i)}(t) = \begin{bmatrix} \frac{1}{\hat{s}_x^{(i)}(t)} & 0 & 0 \\ 0 & \frac{1}{\hat{s}_y^{(i)}(t)} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.5}$$

$$D^{(i)}(t) = \begin{bmatrix} 1 & 0 & -\hat{d}_x^{(i)}(t) \\ 0 & 1 & -\hat{d}_y^{(i)}(t) \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.6}$$

$$R^{(i)}(t) = \begin{bmatrix} \cos(\hat{\alpha}^{(i)}(t)) & \sin(\hat{\alpha}^{(i)}(t)) & 0 \\ -\sin(\hat{\alpha}^{(i)}(t)) & \cos(\hat{\alpha}^{(i)}(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.7}$$

The affine transformation matrix is then given by $A^{(i)}(t) = S^{(i)}(t) \cdot D^{(i)}(t) \cdot R^{(i)}(t)$.

The matrices $A^{(i)}(t)$ can then be used to generate $y^{(i)}(t, \rho_{max}^{(i)})$. First, we use *affine_grid* to calculate a 2D sampling grid $c^{(i)}(t) \in \mathbb{R}^{X \times Y \times 2}$, which maps each output pixel to a continuous input coordinate. Note that $[-1, 1]$ denotes the image edges. Symmetric extrapolation is implemented by setting $c_{sym}^{(i)}(t) = ((c^{(i)}(t) + 1) \bmod 2) - 1$. We then generate the output frame $y^{(i)}(t, \rho_{max}^{(i)})$ by feeding $I^{(i)}$ and $c_{sym}^{(i)}(t)$ into *grid_sample* with cubic interpolation. Note that for efficiency, one can generate $\mu^{(i)}(t, x, y)$ by interpolating $c_{sym}^{(i)}(t)$ directly, and then feeding the resultant value into *grid_sample*.

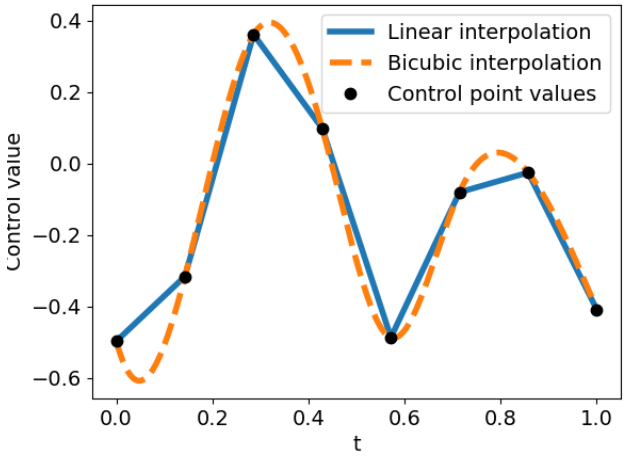


Figure 3.1: Comparison of cubic and linear interpolation of control point values.

3.1.2 Event generation

As stated earlier, the input to the event simulator is in the form of image sequences, see equation 3.2. The last two steps consist of defining the timestamps t_m and the pre-processing function $G^{(i)}$, whose purpose is to help with generalization across different sensors and sensor parameters. We define it to consists of randomly exponentiating the image, then normalizing it, and finally simulating illuminance ranges by scaling it as follows:

$$G^{(i)}(x) = L^{(i)}_{min} + \text{norm}_{[0,1]}(\exp(k^{(i)} \cdot x))(L^{(i)}_{max} - L^{(i)}_{min}), \tag{3.8}$$
$$\tag{3.9}$$

where $L_{min} \sim U[450, 850]$, $L_{max} \sim U[10'000, 15'000]$, and $k \sim U[0.7, 2.5]$. We also define t_m, $m \in \{1, ..., M\}$ as

$$t_m = \frac{m-1}{M-1}. \tag{3.10}$$

Finally, we use the parameters in table 3.1 to generate the training and test datasets.

Table 3.1: Data generation parameters used for the training and test datasets. These values were chosen empirically by qualitatively comparing the generated sequences to the E2VID dataset. To exemplify these values, $\alpha = 2\pi$ represents a full 360° rotation of the image, $s_x = 2$ represents a 2x horizontal "zoom-in", and $t_x = 1$ represents translating the image center to its right edge. The test dataset contains slightly longer and more challenging sequences.

	M	N_{max}	s_{max}	α_{max}	d_{max}	S_{max}
Train	1000	10	0.7	1.7	0.6	4
Test	1500	15	1	2	0.75	6

3.2 Event Representation

Before events ϵ can be processed by a RCNN, they have to be converted to a fixed-size tensor E. We use the standard approach of encoding events into a spatiotemporal voxel grid [3]. The input events are split into T temporal *frames*, each corresponding to some timespan $(t_{\tau-1}, t_\tau)$. Note that this timespan can be of different length for each frame τ. For example, E2VID proposes creating frames such that the number of events in each frame is constant. In real datasets where APS frames are available, the event frames are usually fixed such that their edges coincide with the APS frames. In our approach, we use frames of a fixed temporal width, such that $t_\tau - t_{\tau-1}$ is constant. For use $T_{train} = 100$ and $T_{test} = 150$ the the training and test datasets.

Each frame is then further split into B temporal *bins* of width

$$\delta t_{\tau,b} = \frac{t_\tau - t_{\tau-1}}{B}, \tag{3.11}$$

such that each bin $b \in \{1, ..., B\}$ corresponds to the time interval

$$I_{\tau,b} = [t_{\tau-1} + (b-1)t_{\tau,b}, t_{\tau-1} + bt_{\tau,b}]. \tag{3.12}$$

This allows us to form voxel grids of dimensionality $\mathbb{R}^{T\times B\times X\times Y}$, where each voxel index (τ, b, x, y) corresponds to the events

$$S_{\tau,b,x,y} = \{e_i \in \epsilon | x_i = x, y_i = y, t_i \in I_{\tau,b}\}. \tag{3.13}$$

Then, we form four voxel grids as follows:

$$E_{polarity}(\tau, b, x, y) = \sum_{i:e_i \in S_{\tau,b,x,y}} p_i, \tag{3.14}$$

$$E_{mean}(\tau, b, x, y) = \frac{1}{|S_{\tau,b,x,y}|} \sum_{i:e_i \in S_{\tau,b,x,y}} \hat{f}_{\tau,b}(t_i), \tag{3.15}$$

$$E_{std}(\tau, b, x, y) = \sqrt{\frac{1}{|S_{\tau,b,x,y}|} \sum_{i:e_i \in S_{\tau,b,x,y}} (\hat{f}_{\tau,b}(t_i) - E_{mean}(\tau, b, x, y))^2}, \tag{3.16}$$

$$E_{count}(\tau, b, x, y) = |S_{\tau,b,x,y}|, \tag{3.17}$$

where

$$\hat{f}_{\tau,b}(t) = \frac{t - (t_{\tau-1} + (b-1)t_{\tau,b})}{t_{\tau,b}} \tag{3.18}$$

normalizes the timestamp to $[0, 1]$ within a given frame τ and bin b. Note that if $S_{\tau,b,x,y}$ is empty, we set the corresponding values to 0. Semantically, E_{mean} is the average normalized timestamp of events in a voxel, E_{std} is the standard deviation, and E_{count} is the number of such events. We then form the final tensor $E \in \mathbb{R}^{T\times 4B\times X\times Y}$ by concatenating these voxel grids along the second dimension.

The standard approach (e.g. [3], [24]) is to only include information about the polarity within a bin, similar to our $E_{polarity}$. However, in these methods, the neural

network only outputs the intensity at the end of every frame. In our approach, we can query the intensity at any timestamp t, so we benefit from the additional temporal information provided by E_{mean} and E_{std}. Our experiments confirm this, and also show that including E_{count} is beneficial. Note that the discretization step is much faster than running the neural network, so we lose little efficiency by including this extra information.

3.3 Continuous Video Representation

As opposed to the approach presented in VideoINR [5], our representation treats the temporal dimension similarly to the spatial dimensions. It can be seen as a straightforward extension of LIIF [4] from the 2D spatial case to the 3D spatiotemporal case. Therefore, we attempt to closely follow their notation and derivation here. We represent each continuous video $V^{(i)}$ as a 3D feature map $M^{(i)} \in \mathbb{R}^{T \times X \times Y \times D}$. All videos share a decoding function C_ψ which takes the form

$$s = C_\psi(z, \mathbf{x}), \tag{3.19}$$

where $z \in \mathbb{R}^D$, $\mathbf{x} \in \mathbb{R}^3$ is a 3D continuous video coordinate, and $s \in \mathbb{S}$ is the predicted signal. In our case, s consists of two scalars - the intensity and the variance, so $S = \mathbb{R}^2$, and the range of $\mathbf{x}$ is $[0, T]$, $[0, X-1]$, and $[0, Y-1]$. Fixing C_ψ, a vector z can then be seen as representing a function $C_\psi(z, \cdot) : \mathbb{R}^3 \to \mathbb{S}$. Next, we assign evenly distributed coordinates v to the $T \times X \times Y$ latent codes in $M^{(i)}$ as follows:

$$v_{t,x,y} = \begin{bmatrix} t-1 \\ x-1 \\ y-1 \end{bmatrix}. \tag{3.20}$$

Then, the predicted values at continuous coordinate $\mathbf{x}_q$ are defined by

$$V^{(i)}(\mathbf{x}_q) = C_\psi(z^*, \mathbf{x}_q - v^*), \tag{3.21}$$

where z^* is the latent code nearest to $\mathbf{x}_q$, and v^* is the coordinate of z^*. For example, in figure 3.2, $z^* = z^*_{100}$, and v^* is its coordinate. Thus, each latent code z in $M^{(i)}$ represents a volume of the continuous video surrounding its coordinates, and is responsible for predicting the signal for coordinates in that volume.

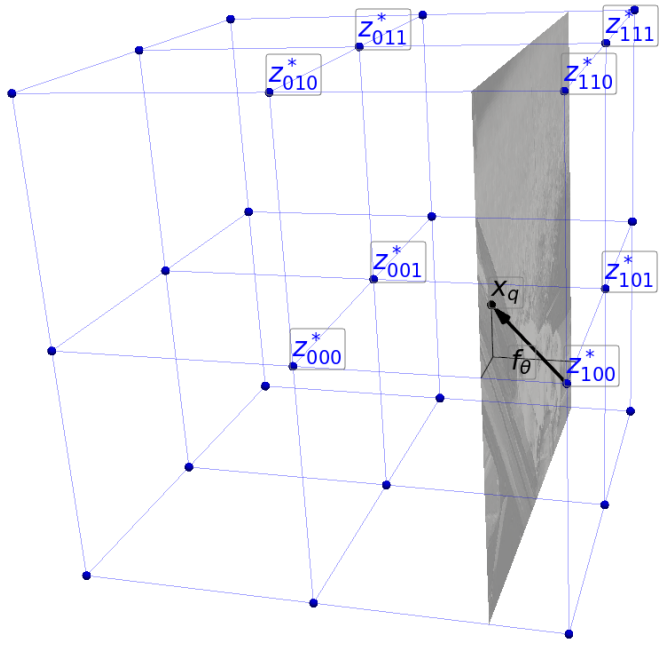


Figure 3.2: Following LIIF [4], a video is represented as a 3D feature map and a function C_ψ. The final output is predicted by trilinearly interpolating neighboring predictions.

Continuing to follow LIIF, we also employ feature unfolding and prediction interpolation (which they call local ensembles).

Feature unfolding refers to the concatenation of the $3 \times 3 \times 3$ neighboring latent codes in $M^{(i)}$ to form $\hat{M}^{(i)}$:

$$\hat{M}^{(i)}_{t,h,w} = \text{Concat}(\{M_{t+i,h+j,w+k}\}_{i,j,k\in\{-1,0,1\}}), \tag{3.22}$$

where $M^{(i)}$ is zero-padded. Following feature unfolding, $\hat{M}^{(i)}$ replaces $M^{(i)}$ for all computations. This is similar to a 3×3 convolutional kernel, and the idea is that the additional spatial information is helpful when decoding the latent code.

Prediction interpolation addresses a problem in equation 3.21, namely the possibility of discontinuities in the predicted signal. Specifically, there exist planes where the selection of the nearest latent code switches, and the predictions for two infinitesimally close coordinates can be different as long as $M^{(i)}$ or C_ψ are not perfect. In practice, this effect is especially noticeable in the temporal dimension, as the predicted videos show a discontinuous jump when the latent code switches.

To circumvent this, we predict the signal at $\mathbf{x}_q$ using the surrounding eight latent codes z^*_{thw}, where $t, h, w \in \{0, 1\}$, see figure 3.2. Then, we assign each prediction the coordinates of the corresponding latent code, and use trilinear interpolation to compute the final prediction for $\mathbf{x}_q$. This method makes the transition between nearest latent codes z^* smooth, and the predictions continuous. For example, the

prediction for a coordinate exactly in the middle between two latent codes is the average of the two predictions generated by them.

Note that with the latent vector coordinates defined in 3.20 and the given range of $\mathbf{x}$, all coordinates $\mathbf{x}$ are spatially surrounded by latent codes. However, the temporal range of $\mathbf{x}$ is $[0, T]$, whereas the largest latent code temporal coordinate is $T - 1$. Therefore, for $\mathbf{x}$ such that $\mathbf{x}_0 \geq T - 1$, we only use the four latent codes z^*_{0hw}, and use bilinear interpolation to get the final prediction.

3.4 Network Architecture and Training

Our method employs two neural networks; an RCNN ν_ϕ, which encodes event tensors $E \in \mathbb{R}^{T \times 4B \times X \times Y}$ into feature maps $M \in \mathbb{R}^{T \times X \times Y \times D}$, and an MLP C_ψ, which decodes unfolded latent codes $z \in \mathbb{R}^{27D}$ and relative coordinates $\mathbf{x}_{rel} \in \mathbb{R}^3_{[-1,1]}$ into predicted intensities and variances $\hat{y} \in \mathbb{R}^2$. It is a small model with 4 hidden layers and a hidden size of 128.

The architecture of ν_ϕ is inspired by the two most popular event to image reconstruction architectures - E2VID [3] and FireNet [22]. The exact setup is visualized in figure 3.3. Like FireNet, we replace the ConvLSTM layers in the E2VID architecture with the more efficient ConvGRU [35] layers. We also use instance normalization [36] instead of batch normalization and Exponential Linear Units (ELU) [37] instead of ReLUs.

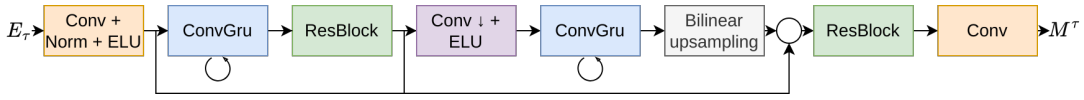


Figure 3.3: Visualization of the architecture we use for ν_ϕ. We use kernel size 3 and hidden size 24 for all convolutional layers. *Conv* refers to a convolutional layer with stride 1 and *Conv*↓ is a convolutional layer with stride 2. The empty circle represents concatenation. The *ResBlock* is visualized in figure 3.4.

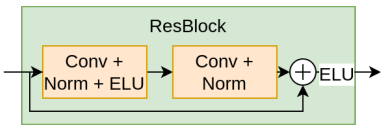


Figure 3.4: Visualization of the residual block (ResBlock) we use in our architecture. The circle with a plus represents addition.

3.4.1 Loss

Simply using $\mathcal{L}_{MVE}$ as shown in equation 2.7 leads to low quality reconstructions. This is because with a fixed $\hat{\sigma}$, the function $\mathcal{L}_{MVE} \sim (\mu_i - \hat{\mu}_i)^2$, which is simply per-pixel mean squared error (MSE). Johnson et al showed that replacing MSE with a perceptual loss function leads to significantly better performance in image

transformation tasks [38]. Instead of minimizing the per-pixel difference between the prediction and the target, perceptual losses instead aim to minimize the difference in features extracted by passing the prediction and target images through pre-trained CNNs. We define our loss function as

$$\mathcal{L}(\hat{\mu}, \hat{\sigma}, \mu) = \lambda_{MVE}\mathcal{L}_{MVE}(\hat{\mu}, \hat{\sigma}) + \mathcal{L}_{LPIPS\_VGG}(\hat{\mu}), \tag{3.23}$$

where $\mathcal{L}_{LPIPS\_VGG}$ is the Learned Perceptual Image Patch Similarity metric based on the VGG network [39]. By comparing the value ranges of the two loss functions, we find that $\lambda_{MVE} = 0.2$ works well.

Furthermore, we do not back-propagate the gradients of $\mathcal{L}_{MVE}$ through $\hat{\mu}$. The rationale behind this is two-fold:

1. MVE networks are notorious for convergence difficulties due to the coupling of the mean and variance [40]. Our approach lets us naturally decouple the two, leading to stable training.

2. Without this gradient manipulation, our loss function could be seen as a weighted sum of LPIPS and MSE, whereas [38] showed that purely perceptual losses lead to better results.

Finally, to avoid issues with predicted negative standard deviations, we define the network output as $\hat{\sigma}' = \ln \hat{\sigma}$. Thus, the final loss function becomes

$$\mathcal{L}_{final}(\hat{\mu}, \hat{\sigma}', \mu) = \lambda_{MVE}\mathcal{L}_{MVE}(\text{stop\_grad}[\hat{\mu}], \exp[\hat{\sigma}']) + \mathcal{L}_{LPIPS\_VGG}(\hat{\mu}). \tag{3.24}$$

3.4.2 Training procedure

Our optimization problem can be formulated as follows

$$\min_{\phi,\psi} \sum_i \sum_{\tau \leq T} \mathcal{L}_{final}\Big(C_\psi(\nu_\phi(E^{(i)}), \tau + \delta t_\tau - 1, \rho^{(i)}_{min} \cdot s), \quad y^{(i)}(\frac{\tau + \delta t_\tau - 1}{T}, \rho^{(i)}_{min} \cdot s)\Big), \tag{3.25}$$

where $\delta t_\tau \sim U[0, 1]$ and $s \sim U[1, 4]$. Note that the term $\frac{\tau+\delta t_\tau - 1}{T}$ is a remapping from the domain of $\tau + \delta t_\tau$ ($[1, T+1]$) to the domain used to generate ground truths ($[0, 1]$). By $\nu_\phi(E^{(i)}_\tau)$, we denote the feature map $M^{(i)}$ of shape generated by encoding the event tensor $E^{(i)}$ with the encoder ν_ϕ. By $C_\psi(M^{(i)}, t, \rho)$, we denote the image generated from the feature map $M^{(i)}$ at resolution ρ and timestamp $t \in [0, T]$.

The exact training procedure is described in algorithm 1. For simplicity, we omit batches in the algorithm, but the networks are trained with batch size 2. Note that the loop over the pixels in $y_{\tau,crop}$ is vectorized, which significantly improves training efficiency at the cost of GPU memory usage. We use $N_{iters} = 3 \cdot 10^5$, and optimize the network weights using the Adam optimizer [41] with learning rate $\gamma = 0.0005$ and other settings left at their *Pytorch* defaults. Training takes around 72 hours on one NVIDIA RTX 6000 ADA 48 GB. The batch and crop sizes were chosen such that they fit on the GPU.

Note that even though our representation treats the spatial dimensions very similarly to the temporal dimensions, our training algorithm treats them completely differently. This is due to the way we generate ground-truths. We first generate the transformation matrix $A(t)$, which is then used to generate a grid mapping each output pixel to an input coordinate. This is computationally expensive, so in our training algorithm, we only do it once per event frame. While one could generate a set of fully independent spatiotemporal coordinates, it would be significantly less efficient, as well as harder to implement, visualize, and debug. However, we suspect that constraining the temporal coordinate in this way might be contributing to our very long training times, and further investigation is needed.

Algorithm 1 Training algorithm

Input: Event tensors $E^{(i)}$, ground-truth generating functions $y^{(i)}$, RCNN ν_ϕ, and MLP C_ψ
Output: Trained network weights ψ and ϕ
for $iter = 1$ **to** N_{iters} **do**
    sample $i \sim U\{1, 500\}$
    sample a frame index $\tau_{start} \sim U\{1, T - 15\}$
    sample a spatiotemporal crop $E \in \mathbb{R}^{16\times 4B\times 56\times 56}$ from $E^{(i)}$, with the first frame corresponding to τ_{start}
    sample prediction resolution $\rho \sim U\{\rho^{(i)}_{min,1}, \rho^{(i)}_{max,1}\} \times U\{\rho^{(i)}_{min,2}, \rho^{(i)}_{max,2}\}$.
    encode E using ν_ϕ to get $M_{iter} \in \mathbb{R}^{16\times D\times 56\times 56}$
    loss = 0
    for $\tau = 1$ **to** 16 **do**
        sample prediction timestamp $t_\tau = (\tau_{start} - 2 + \tau + \delta t)/T$, where $\delta t \sim U[0, 1]$
        generate $y_\tau = y^{(i)}\left(t_{\tau,iter}, \rho^{(i)}_{max}\right)$
        interpolate y_τ at spatial coordinates corresponding to the crop in E at resolution ρ, generating $y_{\tau,crop}$
        for each pixel x, y in $y_{\tau,crop}$ **do**
            predict $\hat{\mu}_{x,y}$ and $\hat{\sigma}'_{x,y}$ from M_{iter} using C_ψ, see section 3.3
            loss $+= \mathcal{L}_{final}(\hat{\mu}_{x,y}, \hat{\sigma}'_{x,y})$
        end for
    end for
    backpropagate loss and update weights ψ, ϕ using Adam optimizer
end for

3.5 Evaluation

Following academic standards in image reconstruction from events [25], we use three metrics for evaluation of our reconstructions: per-pixel Mean Squared Error (MSE), LPIPS (AlexNet), and Structured Similarity Index Measure (SSIM).

For evaluating uncertainty estimations, we begin by employing two standard metrics [42]:

1. 80% Prediction Interval Coverage Probability (PICP80) - the fraction of pixel intensities μ in the test set that fall within the 80% prediction interval, which given our distribution $N(\hat{\mu}, \exp(\hat{\sigma'}))$ is equal to [43]:

$$PI_{80\%}(\hat{\mu}, \hat{\sigma'}) = [\hat{\mu} - 1.28 \exp(\hat{\sigma'}), \hat{\mu} + 1.28 \exp(\hat{\sigma'})] \,. \tag{3.26}$$

2. Loglikelihood (LL) - the logarithmic likelihood of the test data, which in our case can be calculated as

$$\begin{aligned} LL(\hat{\mu}, \hat{\sigma'}) &= \ln \left[\frac{1}{\sqrt{2\pi \exp(\hat{\sigma'})^2}} \exp\left(-\frac{1}{2} \left(\frac{\mu - \hat{\mu}}{\exp(\hat{\sigma'})} \right)^2 \right) \right] \\ &= -\frac{\ln(2\pi)}{2} - \hat{\sigma'} - \frac{1}{2} \left(\frac{\mu - \hat{\mu}}{\exp \hat{\sigma'}} \right)^2 . \end{aligned} \tag{3.27}$$

These two methods have their drawbacks - PICP can be misleading, as simply always predicting very large variances would lead to very good scores, whereas LL values can be difficult to interpret. Furthermore, these values measure the difference between μ and $\hat{\mu}$ simply as $(\mu - \hat{\mu})^2$, ignoring perceptual context. Therefore, we propose a third metric:

3. LPIPS and Standard Deviation Correlation (LPIPS_CORR) - per-patch correlation between LPIPS and median predicted standard deviation over the test data. This metric is meant to describe how well the network predicts the perceptual difference between the ground truth and the predicted reconstruction. Its utility is explained in section 4.1.1.2.

4

Results

In this chapter, we present quantitative and qualitative results. We begin by evaluating our method on the simulated test dataset, showing performance on prediction, spatial superresolution, frame interpolation, and uncertainty quantification. Then, we move on to real data, applying our method to the Event Camera Dataset (ECD) [30] and the High Quality Frames (HQF) dataset [23]. We compare our intensity predictions to several established RCNN-based algorithms such as E2VID [3], FireNET [22], FireNET+, and E2VID+ [23]. Then, we evaluate the uncertainty predictions, and show qualitative results on superresolution and frame interpolation.

4.1 Simulated data

In this section, we show results on our simulated test dataset.

4.1.1 Prediction without superresolution

We begin by evaluating predictions without any spatiotemporal superresolution, i.e. given an event tensor $E \in \mathbb{R}^{T\times 4B\times X\times Y}$, we predict an output $\hat{y} \in \mathbb{R}^{T\times X\times Y\times 2}$. Evaluated on our test dataset, we get the results shown in table 4.1.

Table 4.1: Results on the test dataset using prediction without superresolution or temporal interpolation.

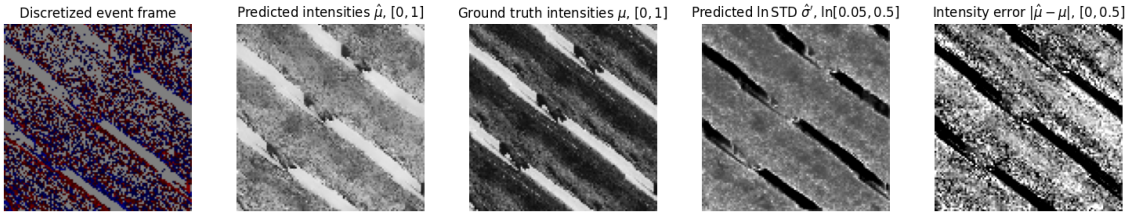
MSE	LPIPS (Alex)	SSIM	PICP80	LL	LPIPS_CORR
0.018 ± 0.012	0.13 ± 0.05	0.61 ± 0.13	0.79 ± 0.12	0.76 ± 0.35	0.19 ± 0.2

While one could easily apply other SOTA intensity prediction methods to our test dataset, we elect not to show these results. Because our method is trained on data that is highly similar to the test data, no conclusions could be drawn from such a comparison. Instead, we compare our method to others only on real data, and the values above mainly serve as a baseline.

4.1.1.1 Intensity predictions

To begin with, we analyze our intensity predictions $\hat{\mu}$, beginning by showing examples. First, we highlight some low performance cases with LPIPS (Alex) far worse than the average in figure 4.1. Then, we show examples of predictions with LPIPS (Alex) close to average in figure 4.2.

(a) LPIPS (Alex) = 0.292. With too many events, the method fails to reconstruct fine details, such as the giraffe coat patterns.



(b) LPIPS (Alex) = 0.386. Due to the inherent sparsity of event data, the method can fail to correctly reconstruct the brightness of the image.

(c) LPIPS (Alex) = 0.237. Example of failure due to a lack of events.

Figure 4.1: Examples of predictions generated by our networks with LPIPS (Alex) far above the average, with short explanations for the bad performance.

(a) LPIPS (Alex) = 0.13.

(b) LPIPS (Alex) = 0.14.

Figure 4.2: Examples of predictions generated by our networks with LPIPS (Alex) close to the average.

The representative examples from figure 4.2 show that our method generates high-quality reconstructions, generally capturing both fine details and overall image structure. However, the examples in figures 4.1c and 4.1a indicate that intensity prediction performance can be sensitive to the number of events in the event frames. To gain a deeper understanding of this relationship, we plot a heatmap of LPIPS and event counts in figure 4.3. We normalize the event counts by the image resolution, thus forming the **EPF** (Event per Pixel per Frame) metric.

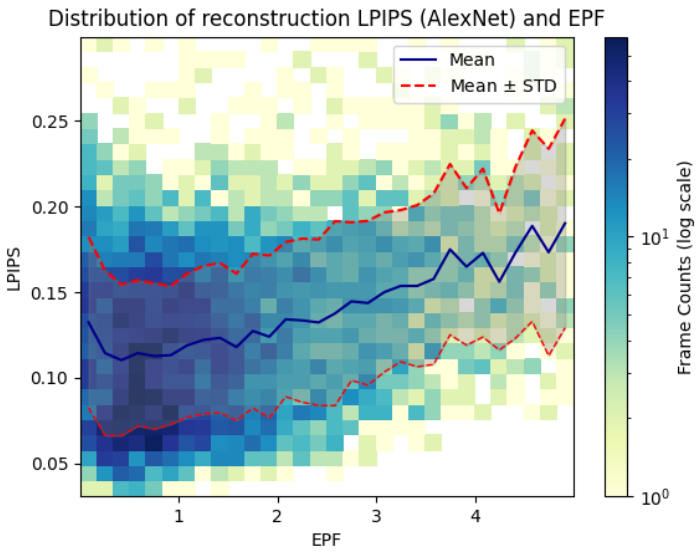


Figure 4.3: 2D heatmap of LPIPS versus EPF over all frames in the test dataset, with predictions generated without superresolution or interpolation. Data points are aggregated into 50 bins per dimension, where each bin represents a specific range of LPIPS and EPF values. Note that LPIPS is clipped to $[0, 0.3]$ and EPF to $[0, 5]$. Additionally, statistical lines are plotted using 1D binning along the EPF axis. For each EPF bin, we show the mean (blue line) and standard deviation (dashed red line) of the LPIPS values of patches within that bin.

While figure 4.3 shows the expected increase in LPIPS values for very low and very high EPF, the general tendency is that LPIPS increases with EPF. This is surprising; intuitively, it seems that more events would give the networks more information, and lead to more accurate representations. However, note that EPF depends on three main factors:

1. sensor sensitivity S, determined by the sensor parameters and sensor noise, and further augmented by our preprocessing function G,

2. scene characteristics C, such as brightness (lower brightness leads to more events), contrast, or level of detail,

3. magnitude of motion M, determined by the affine matrix generating function $A(t)$.

While higher S increases the number of events without impacting the inherent difficulty of intensity prediction, C influences both the number of events and the difficulty. Additionally, increasing M generates more events at the cost of increasing the difficulty of intensity prediction, and figure 4.3 indicates that this is the dominating factor. While it would be informative to analyze the relationship between S and LPIPS, S cannot readily be measured, making such analysis difficult.

4.1.1.2 Uncertainty predictions

Next, we analyze the uncertainty predictions $\hat{\sigma}$. We begin by showcasing the complementary roles of the three metrics (LL, PICP80, and LPIPS_CORR) used to evaluate the uncertainty predictions. In figure 4.5, we show examples of predictions with combinations of high and low LPIPS_CORR, PICP80, and LL values. High LL and PICP80 values suggest that the predicted $\hat{\sigma}'$ is proportional to the MSE, whereas high LPIPS_CORR indicates that it corresponds to the perceptual difference between the predicted image and the ground truth. Note that the network is not trained to estimate the perceptual difference, so examples with high LPIPS_CORR and low LL or PICP80 are rare.

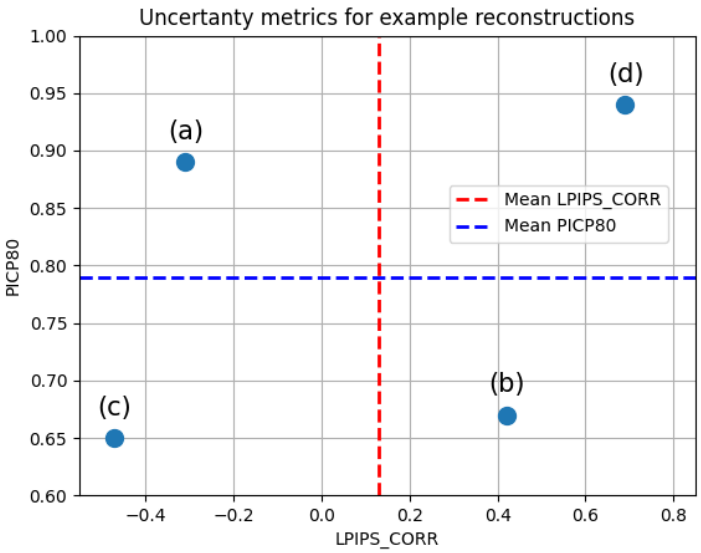
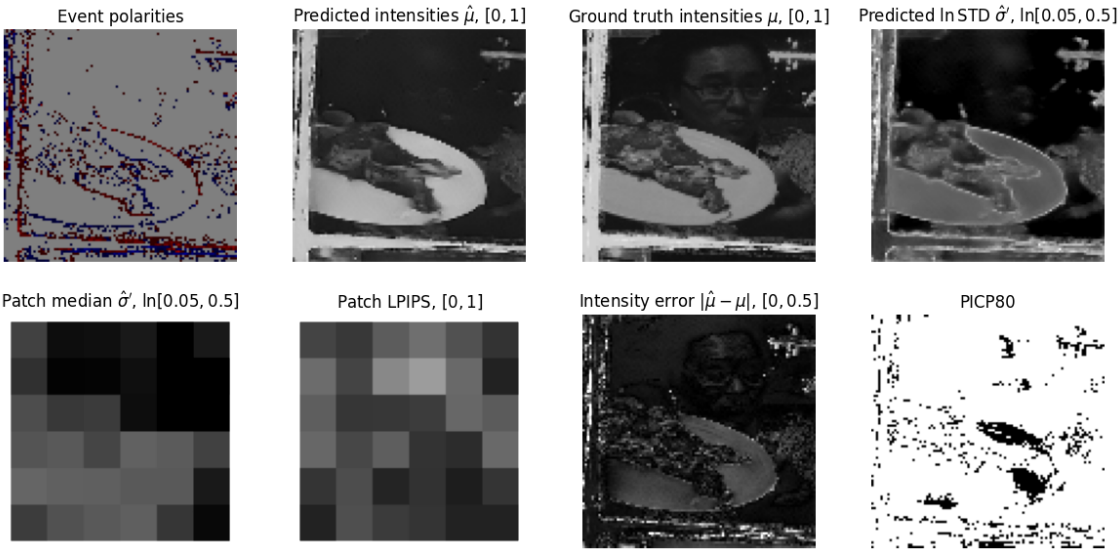
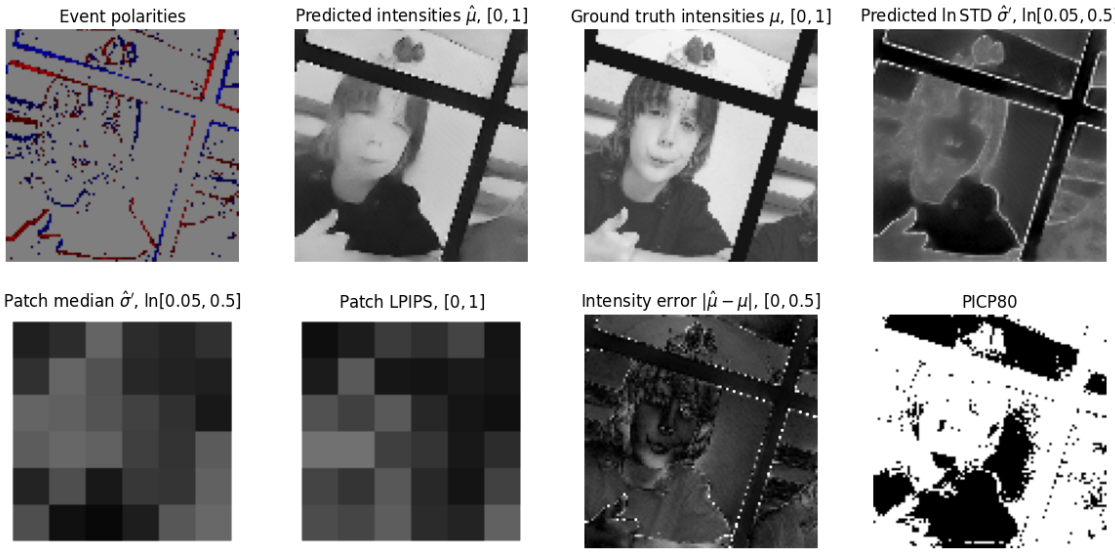


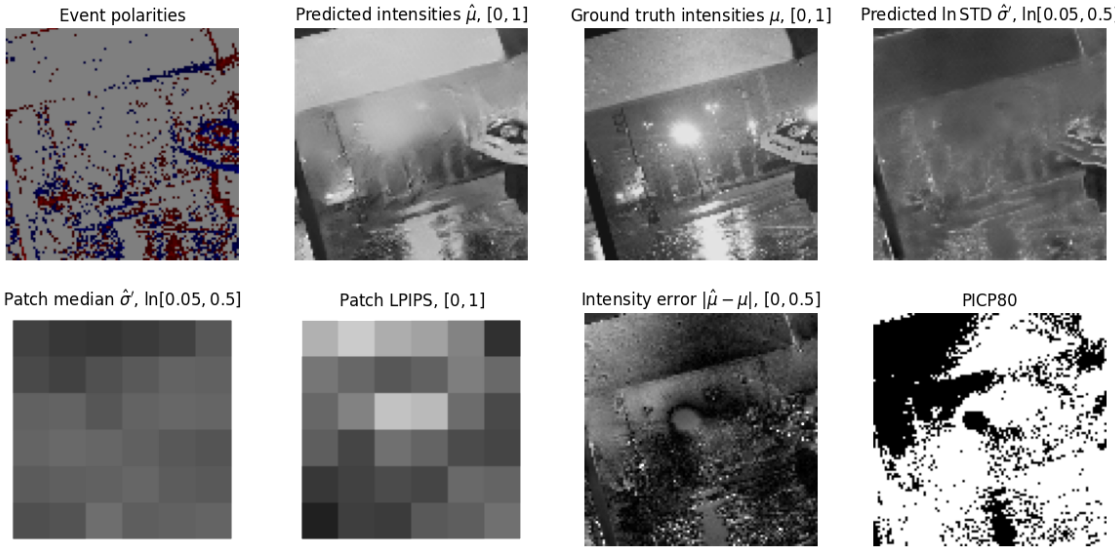
Figure 4.4: Distribution of the LPIPS_CORR and PICP80 values of the example predictions shown in figure 4.5.



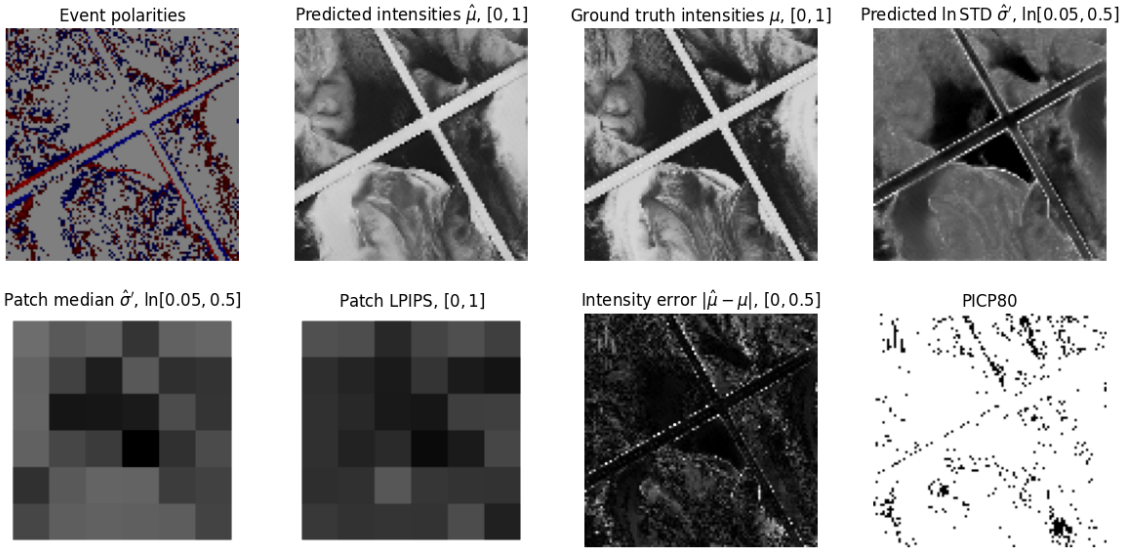
(a) High LL = 1.2 and PICP80 = 0.89 with low LPIPS_CORR = −0.31. The predicted $\hat{\sigma}'$ correctly identifies areas with high MSE, leading to high LL and PICP80 values. However, note that the face is barely reconstructed, and therefore has high LPIPS values despite low MSE values. On the other hand, the plate has high $\hat{\sigma}$ and high MSE, even though it is perceptually reconstructed well. These two areas explain the negative LPIPS_CORR.



(b) Low LL = 0.66 and PICP80 = 0.67 with high LPIPS_CORR = 0.42. The network attributes a high uncertainty to the facial area, which has a low MSE, resulting in a low LL. However, the intensity predictions in this area have low perceptual quality, and this can be seen in the patch LPIPS values, resulting in a high LPIPS_CORR. Furthermore, the high confidences assigned to the incorrectly predicted shirt and background colors result in a low PICP80, despite these areas being perceptually well-reconstructed.



(c) Low LL = 0.36, PICP80 = 0.65, and LPIPS_CORR = -0.47. The predicted $\hat{\sigma}'$ does not identify erroneous areas such as the lamp or the top part of the image.



(d) High LL = 1.14, PICP80 = 0.94, and LPIPS_CORR = 0.69. The predicted $\hat{\sigma}'$ identifies both well reconstructed areas (image center) and erroneous areas (bottom center, center right).

Figure 4.5: Examples showcasing the complementary roles of the novel LPIPS_CORR and traditional LL and PICP80 uncertainty quantification metrics. The value ranges in the figure titles represent colors from black to white. For PICP80 plots, white pixels indicate μ is within the prediction interval.

The examples from figure 4.5 suggest that if images are reconstructed in the context of downstream computer vision applications, the novel LPIPS_CORR score can arguably be more important than LL or PICP80. For example, consider applying a face detection algorithm to our intensity predictions. In figure 4.5a, the face is barely reconstructed at all, and yet the predicted $\hat{\sigma}'$ in this area is very low, resulting in a negative LPIPS_CORR value. This would be fatal for the face detection algorithm, as it would likely have a high confidence that there is no face in the image, and our predicted uncertainties would further confirm this. In contrast, in figure 4.5b, our predicted uncertainties correctly identify that the face is perceptually not reconstructed well which might be reflected in the performance of the downstream algorithm, even though the intensity error is low in that area.

In order to analyze the relationship between predicted STD and LPIPS beyond just the LPIPS_CORR score, we visualize their distributions in figure 4.6. We identify three critical points, at median$(\hat{\sigma}) \approx 0.07$, median$(\hat{\sigma}) \approx 0.13$ and median$(\hat{\sigma}) \approx 0.2$. For the interval $[0.07, 0.13]$, the curve is relatively flat. At 0.13, we identify a sharp increase, followed by flatness until 0.2. At 0.2 we have an elbow point, followed by a clear positive correlation between median$(\hat{\sigma})$ and LPIPS. In figure 4.7, we show examples of patches with median$(\hat{\sigma})$ close to each critical point. This relationship could be used for a downstream computer vision application. For example, one might apply a simple heuristic, stating that $\hat{\sigma} < 0.13$ does not affect the downstream task, $\hat{\sigma} \in [0.13, 0.2]$ lowers the downstream confidences by a constant factor, and for

$\hat{\sigma} > 0.2$, this factor becomes proportional to the predicted $\hat{\sigma}$.

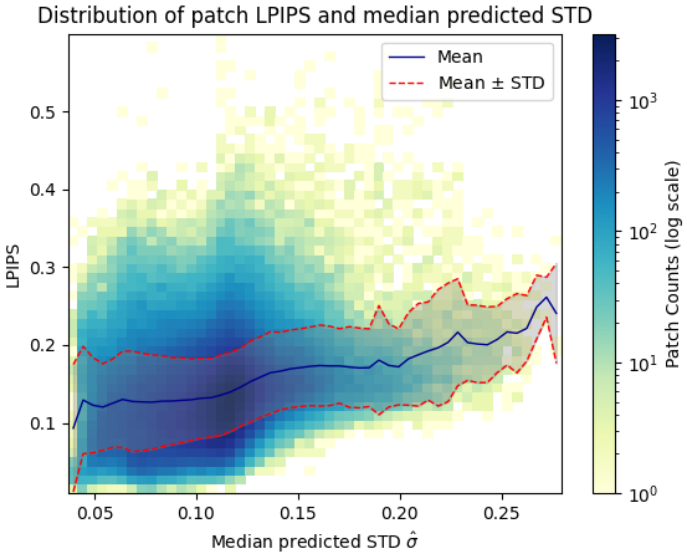


Figure 4.6: 2D heatmap of LPIPS versus Median STD, visualizing the distribution of patch LPIPS values against the median predicted STD over 16 by 16 patches over the test dataset. Data points are aggregated into 50 bins per dimension. Statistical lines were calculated as in figure 4.3.

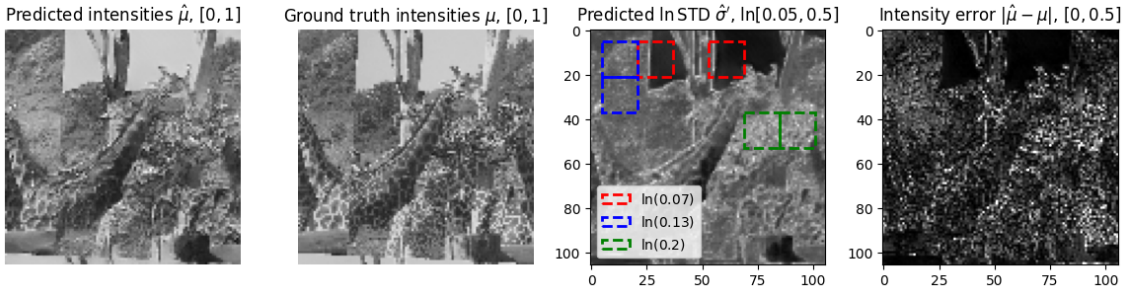


Figure 4.7: Prediction with examples of patches with given median predicted standard deviations highlighted.

The LPIPS_CORR metric has one hyperparameter - the patch size. In figure 4.8, we investigate the influence of the selected patch size on the LPIPS_CORR value over the test dataset. The LPIPS_CORR generally decreases up until a patch size of 32, after which increases. For our evaluation, we use the smallest patch size of 16, as it leads to the smallest variance in LPIPS_CORR.

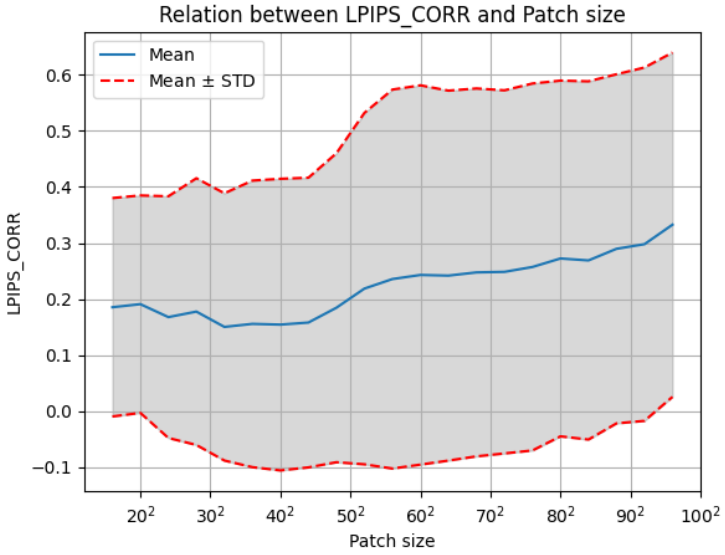


Figure 4.8: Relation between LPIPS_CORR and Patch size. The figure was generated by calculating the LPIPS_CORR over the test dataset for patch sizes between 16^2 and 96^2, with a step of 4. The minimum evaluated patch size is due to LPIPS (AlexNet) needing inputs of size at least 16^2, whereas the maximum is chosen such that it is smaller than the smallest image in the test dataset.

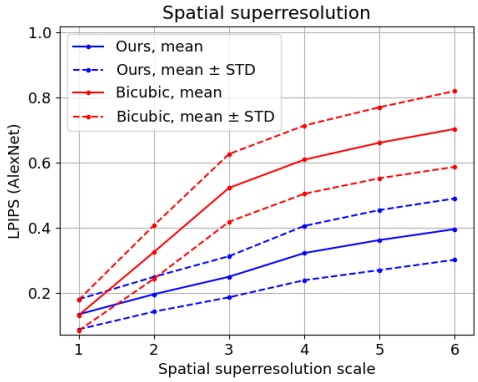
4.1.2 Spatiotemporal superresolution

In the next experiment, we evaluate the spatiotemporal superresolution of our method. We begin by analysing how the perceptual intensity prediction quality changes with temporal and spatial superresolution scale. Then, we do the same for the uncertainty predictions. We conclude by showing examples of predictions generated at different spatiotemporal superresolution scales in figure 4.11.

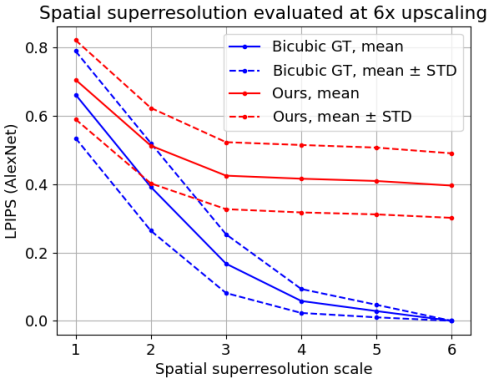
4.1.2.1 Intensity prediction in spatial superresolution

We begin by showing LPIPS as a function of spatial superresolution scale in blue in figure 4.9a. As a baseline, we also show cubic interpolation to the given scale from predictions without superresolution in red.

Figure 4.9a shows that we outperform our baseline, with an LPIPS increase by roughly 200% at 6x superresolution, compared to the $\sim$ 400% of cubic interpolation. However, the LPIPS distance between cubic interpolation and our method remains fairly constant for upscaling factors larger than 3. This might suggest that our method is only better than cubic interpolation up to a cut-off at 3x spatial upscaling, and for bigger scales, the difference is negligible. To confirm this finding, we perform a second experiment, where we first use our method for superresolution by initial spatial scales s, followed by cubic interpolation to 6x spatial upscaling. We calculate LPIPS at 6x upscaling, and plot the values against s in blue in figures 4.9b.



(a) Predictions are generated for spatial scales between 1 and 6 and evaluated against the ground truth. As a baseline, we generate predictions without upscaling and upscale them spatially using cubic interpolation.



(b) Predictions and ground truths are generated for spatial scales between 1 and 6, upscaled to 6x using cubic interpolation, and evaluated against the ground truth.

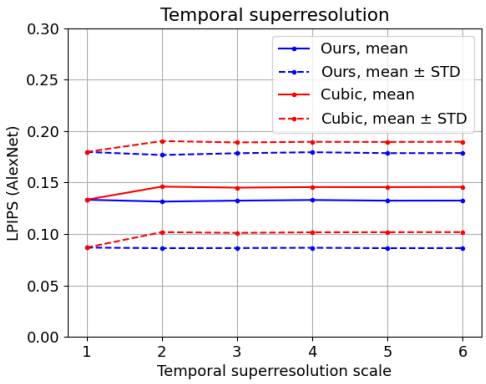
Figure 4.9: LPIPS vs spatial superresolution scale, showing means and standard deviations over all test sequences.

We see an initial decrease in LPIPS followed by a flattening out for scales larger than 3, seemingly confirming our hypothesis. However, this cut-off point does not necessarily have to be intrinsic to our representation, and could be caused by the data itself. For example, consider images of varying resolutions representing a fixed scene. Then, one can choose resolutions large enough that true scene can be approximated arbitrarily well by cubic interpolation. To test if our data has this problem, we generate the ground truth at initial spatial scales s, and then cubically interpolate to 6x spatial upscaling. If the cut-off is intrinsic to the data, we would expect this curve to have a similar flattening for spatial scales larger than 3. If such a flattening does not exist, we can say that the cut-off is intrinsic to the representation. We plot the curve in green in figure 4.9b. It decreases smoothly to 0, indicating that the cut-off scale of 3 is intrinsic to our representation, and not the data.

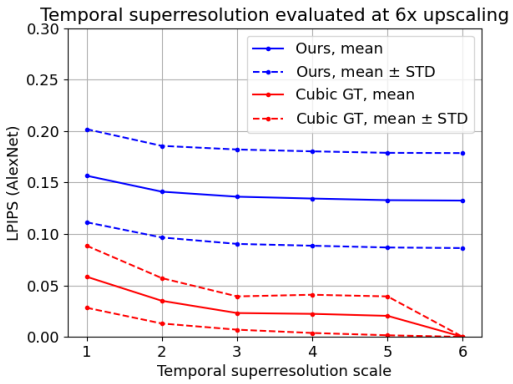
4.1.2.2 Intensity prediction in temporal superresolution

We first show LPIPS as a function of temporal superresolution scale in blue in figure 4.10a. In green, we also show cubic interpolation from predictions without superresolution as a baseline. As opposed to spatial superresolution, where LPIPS decreased significantly as a function of scale, the intensity prediction quality is almost invariant to the temporal superresolution scale. While it is tempting to attribute this to the characteristics of event data, which is spatially discrete and temporally continuous, one must note that the cubic interpolation quality also remains constant for scales larger than 2. This suggests that there is a data cut-off at 2x upscaling after which cubic temporal interpolation approximates true intensities very well. To confirm this, we generate figure 4.10b, the temporal counterpart of figure 4.9b.

While the expected flattening exists, it happens around 3x upscaling, not 2x as expected. However, the prediction curve in figure 4.10b does not have a clear elbow point, and no definitive conclusions about the temporal resolution of our representation can be drawn. To find this value, one would have to generate data with more motion, reducing the temporal consistency of the ground truth. We expect that this would cause the temporal relationships to be more similar to the spatial ones, with more pronounced critical points.



(a) Predictions are generated for temporal scales between 1 and 6 and evaluated against the ground truth. As a baseline, we generate predictions without upscaling and upscale them temporally using cubic interpolation.



(b) Predictions and ground truths are generated for temporal scales between 1 and 6, upscaled to 6x using cubic interpolation, and evaluated against the ground truth.

Figure 4.10: LPIPS vs temporal superresolution scale, showing means and standard deviations of LPIPS over test sequences.

4.1.2.3 Uncertainty quantification in spatiotemporal superresolution

We continue by analyzing the performance of uncertainty quantification as the scale changes. Similarly to figures 4.9b and 4.9b, we generate predictions at spatiotemporal scales between 1 and 4, cubically interpolate them to 4x upscaling, and calculate metrics over the test set, but now we upscale spatially and temporally at the same time. The relationship is similar to what we saw between LPIPS and temporal scale, with a significant increase in performance between scales 1 and 2, followed by a fairly flat relationship.

We do not show performance evaluated against ground truths directly, since the evaluations then use different ground truths and are not comparable. This is especially problematic for LPIPS_CORR, where the perceptual context of a patch is highly dependent on the portion of the scene that it covers. Note that this also explains why the values of LPIPS_CORR shown in table 4.11 are significantly higher than in table 4.1. Generally, it seems that our uncertainty prediction performance is fairly invariant to the upscaling factor. This suggests that while the intensity prediction performance degrades with the scaling factor, the model is able to adapt its uncertainties accordingly.

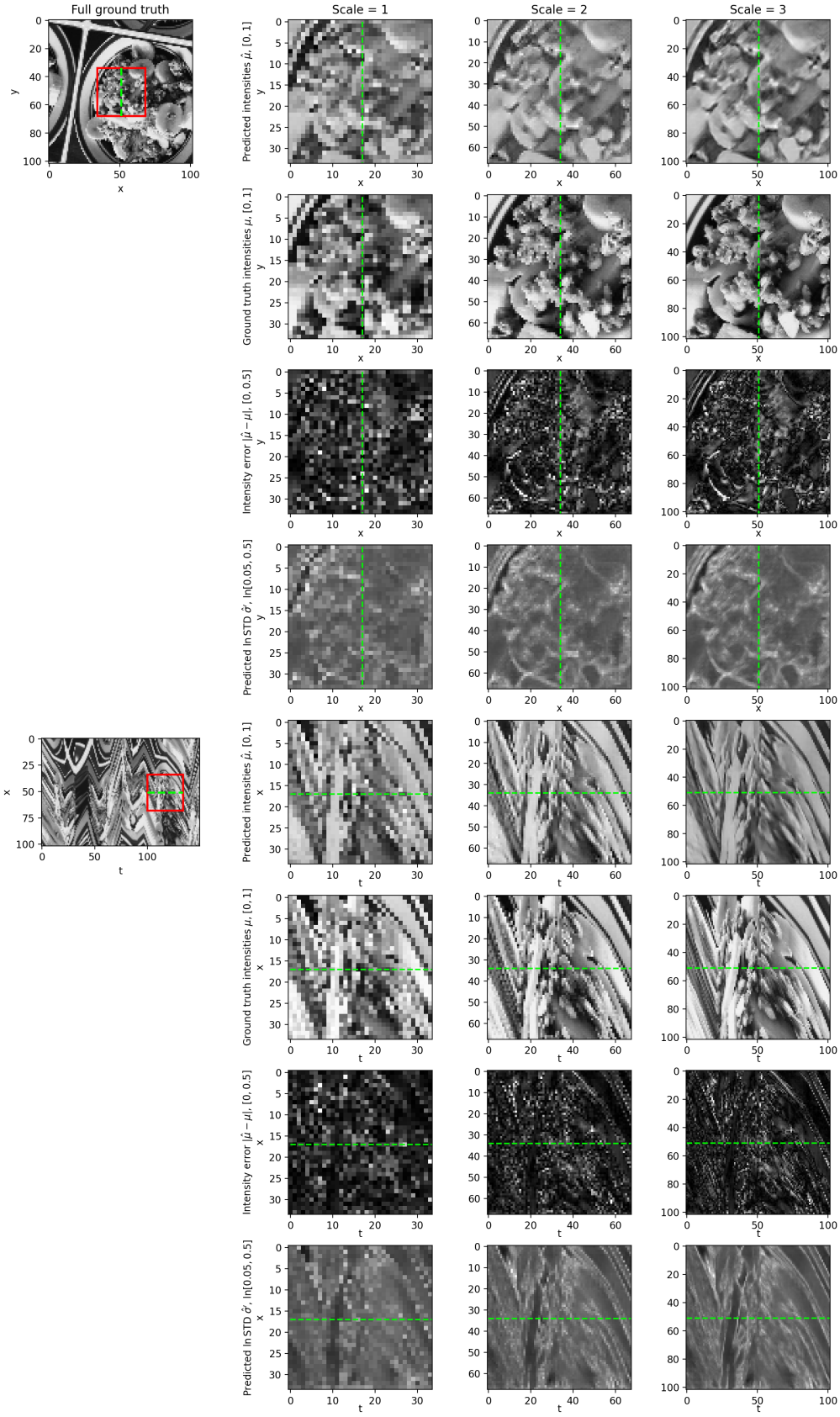


Figure 4.11: Spatiotemporal superresolution example. The x vs t plots have $y = Y/2$, and the x vs y plots have $t = T/2$. The green lines denote $x = X/2$. The red rectangle in the full image is the showcased area.

Table 4.2: Means and standard deviations of key uncertainty metrics over the test dataset calculated by generating predictions at given scales, interpolating them to 4x upscaling, and evaluating against the ground truth. Arrows indicate the direction of better performance for each metric.

Scale	$\bar{\hat{\sigma}}$	MSE ↓	PICP80 ↑	LPIPS_CORR ↑
1	0.11 ± 0.18	0.024 ± 0.012	0.75 ± 0.11	0.37 ± 0.18
2	0.11 ± 0.03	0.017 ± 0.009	0.80 ± 0.11	0.41 ± 0.17
3	0.11 ± 0.03	0.016 ± 0.009	0.81 ± 0.11	0.40 ± 0.18
4	0.11 ± 0.03	0.016 ± 0.009	0.81 ± 0.11	0.40 ± 0.17

4.2 Real data

Finally, we evaluate our method on two real datasets - the Event Camera Dataset (ECD) [30], and the High Quality Frames (HQF) dataset [23]. These datasets contain events and ground-truth intensity images from a DAVIS240C event camera. Note that these intensity images do not necessarily have a fixed temporal frequency. In order to synchronize our reconstructions with the ground truth frames, we form discretized event frames by combining the events between consecutive intensity frames instead of using a fixed temporal width. We begin by providing quantitative results on image reconstruction without any superresolution, and compare them to several other SOTA RCNN-based methods in table 4.3.

Table 4.3: Comparison of our method with several state-of-the-art RCNN reconstruction methods, as reported by Ercan et al. in [25]. LPIPS refers to the AlexNet version here. The best and second best scores are given in **bold** and underlined.

Method	ECD [30]			HQF [23]		
	MSE ↓	SSIM ↑	LPIPS ↓	MSE ↓	SSIM ↑	LPIPS ↓
E2VID [3]	0.179	0.450	0.322	0.099	0.463	0.388
FireNET [22]	0.133	0.459	0.321	0.100	0.422	0.463
E2VID+ [23]	0.070	<u>0.503</u>	**0.236**	**0.036**	**0.536**	**0.255**
FireNET+ [23]	<u>0.062</u>	0.452	0.337	<u>0.045</u>	0.472	<u>0.323</u>
Ours	**0.055**	**0.520**	<u>0.306</u>	0.063	<u>0.479</u>	0.366

In terms of MSE and SSIM, our method surpasses the other methods on ECD, but lags behind both E2VID+ and FireNET+ on HQF. In terms of LPIPS, E2VID+ is better on both datasets, and FireNET+ is better on HQF. Notably, our method was trained on VGG LPIPS, in contrast to the other methods, which were trained on AlexNet LPIPS. This might skew these results in favour of the other methods, as we evaluate on AlexNet LPIPS. In general, we reach performance comparable to E2VID+ and FireNet+, and in figure 4.12, we show examples reinforcing that claim. In order to avoid any bias, we show the same sample scenes as Ercan et al. [25] and Stoffregen et al. [23].

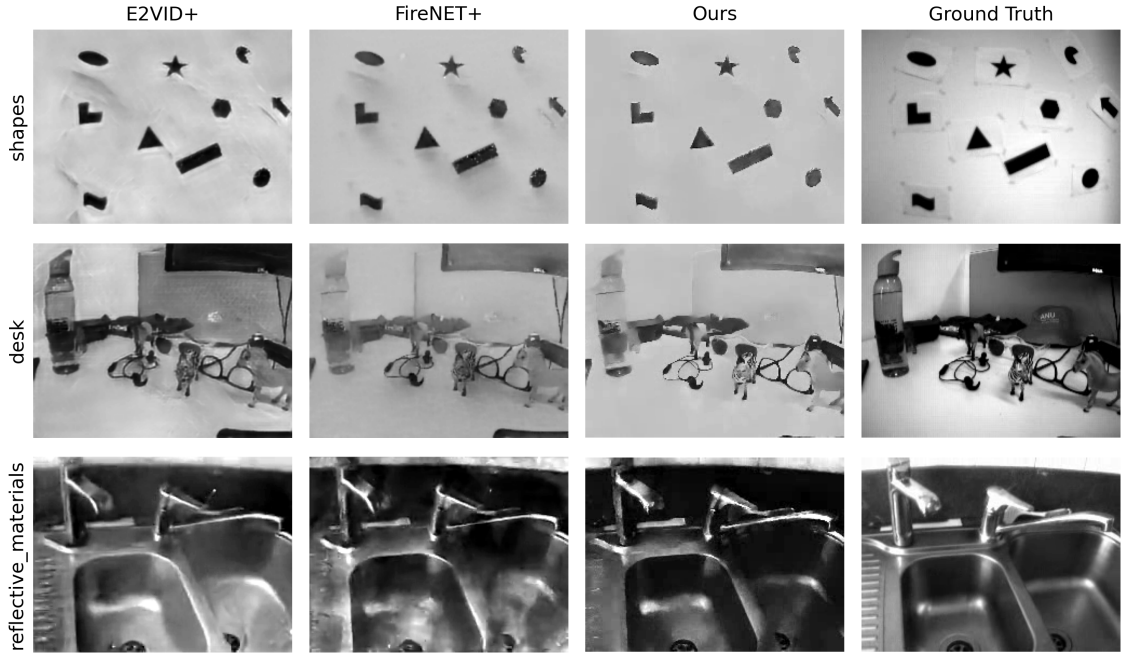


Figure 4.12: Comparison of our method to FireNet+ and E2VID+ on sample scenes from the HQF and ECD datasets.

In the *shapes* scene from the ECD, our method generates fewer artifacts than E2VID+ and FireNET+. However, in the *desk* and *reflective_materials* scenes from the HQF, the intensity of many objects (background, water bottle in *desk*, and overall scene brightness in *reflective_materials*) is significantly worse than in E2VID+. This likely suggests that our simulated data is more similar to the ECD dataset than the HQF dataset, and we discuss this sim-to-real gap in more detail in section 4.2.2.

Evaluating spatiotemporal superresolution quantitatively is difficult, as we do not have access to any superresolved ground-truths. While one could evaluate temporal superresolution by merging neighboring event frames and using the skipped intensity frames as ground truths within the event frame, this would be difficult to interpret, as the task would inherently become harder due to the increase in motion within the frame. Instead, we only evaluate superresolution qualitatively. An example is visualized in figure 4.13, with the patterns on the zebra clearly showing that the superresolution generalizes to real data. However, it remains to be investigated whether this relationship follows the analysis on the simulated data.

Figure 4.13: Spatiotemporal superresolution example on the *desk* scene from the HQF dataset. Note that ground truth intensities are not available for scales larger than 1.

Finally, we show the uncertainty metrics on the two datasets in table 4.4. LL and PICP80 are both drastically lower than on the simulated dataset, suggesting that our uncertainty prediction fails to generalize to real data. However, the average LPIPS_CORR value over the two datasets is surprisingly similar to the simulated data. This indicates that while the predicted uncertainties are a bad estimator of the intensity error on real data, they can still be used to estimate the perceptual error, and could therefore be used for downstream vision applications.

Table 4.4: Uncertainty performance on real data.

ECD			HQF		
LL	LPIPS_CORR	PICP80	LL	LPIPS_CORR	PICP80
-0.73	0.29	0.19	-0.72	0.10	0.23

In many of the analyzed predictions, we have noticed a strong correlation between the brightness of a region and its predicted uncertainties. This is especially obvious in the checkerboard patterns shown in figure 4.14. We have identified three factors that might be contributing to this:

1. Regions with low brightness generate more events, as smaller absolute brightness changes are enough to trigger the Contrast Threshold. While this could enable better predictions, it is important to note that these events are also more noisy, and it is unclear whether the overall effect is positive or negative.

2. Low frequency brightness information is difficult to reconstruct accurately from event data due to its sparsity. For example, in figure 4.14, the brightness of the white cells in the main checkerboard is significantly lower in the bottom right than in the other cells, and the network fails to capture. This low resolution information does not affect the dark cells to the same degree, so the network learns to assign higher uncertainties to bright cells.

3. In the real data, the scene brightness is not static, as movement can cause effects such as reflections and shadows. This is another example of low frequency information which does not affect dark parts of the image as much as bright parts, which can lead to higher uncertainties in bright regions.

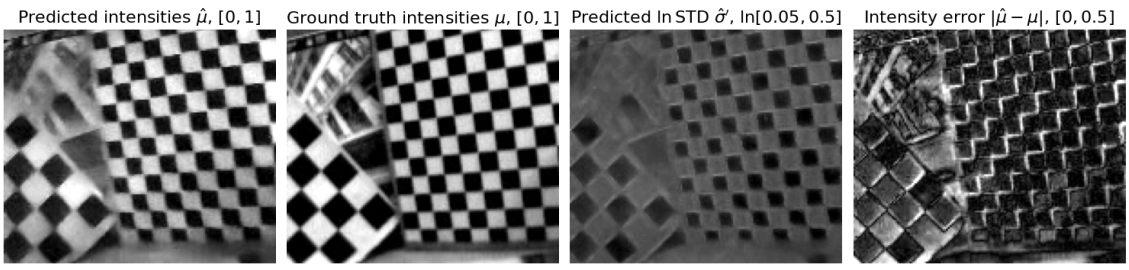


Figure 4.14: Example prediction from the *slow_hand* scene from the HQF dataset.

4.2.1 Performance

Table 4.5 compares the computational performance of our method to FireNET and E2VID. A NVIDIA GeForce RTX 4060 Laptop GPU and an Intel Core i5-12500H were used for all experiments. For FireNET and E2VID, we used the implementation from Scheerlinck et al. [22]. We analyze the performance of the encoder and decoder separately, as the encoder is usually ran at a fixed framerate dependent on the input data, and the decoder determines the output spatiotemporal resolution. On the GPU, our encoder takes roughly 75% more time than FireNET to process an event frame, whereas our decoder takes roughly 40% less time. In figure 4.15, we plot the time it takes to output one second of 240×180) video against the output framerate assuming a fixed event frame duration of 50 ms and fully sequential reconstruction.

It should be noted that this comparison relies on unrealistic assumptions. Mainly, it assumes that the methods run sequentially, whereas in practice, all three are parallelizable, and multiple intensity frames can be decoded simultaneously. Our approach to high-FPS video reconstruction is also very different from the E2VID/FireNET approaches. In their method, the high temporal resolution is achieved by temporally shifting the discretized event frames, whereas in our method, it is intrinsic to the representation. These factors make it hard to draw any conclusion from this comparison, and it should not be interpreted as a claim that our method is faster than FireNET or E2VID.

Table 4.5: Inference time for one event frame on GPU and CPU at the spatial resolutions reported in [23]. Enc and Dec refer to the encoder ν_ϕ and the decoder C_θ.

Resolution	GPU (ms)				CPU (ms)			
	E2VID	FireNET	Enc	Dec	E2VID	FireNET	Enc	Dec
240×180	7.3	2.0	3.7	1.1	120	16	31	11
346×260	18	4.3	7.2	2.4	185	37	79	29
640×480	44	17	31	8.9	640	160	290	94
1280×720	220	60	100	41	2000	640	1200	420

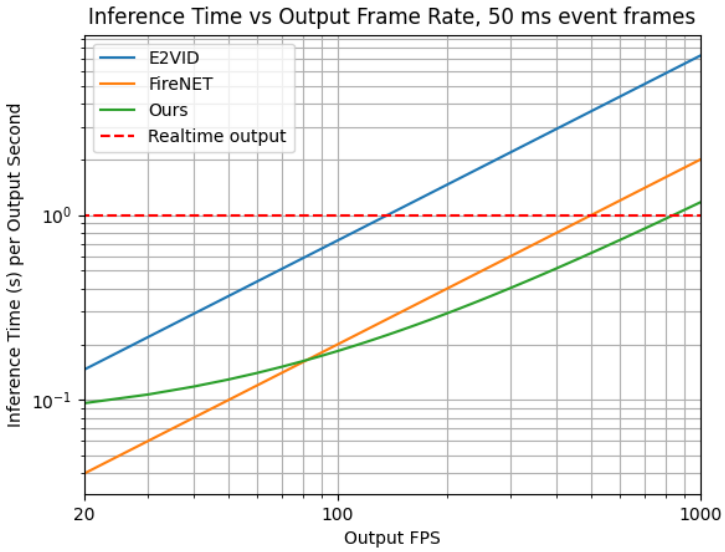


Figure 4.15: The inference time required to generate one second of video output at various frame rates using E2VID, FireNET, and our method. The red dashed line indicates the threshold for real-time inference.

4.2.2 Sim-to-real gap

We have tried several strategies to improve prediction quality on HQF and ECD, including increasing encoder size, replacing our encoder architecture with the E2VID architecture, increasing decoder size, adding data normalization, and training on longer sequences, but none of these attempts led to a significant improvement. Therefore, we believe that the key to improving our performance is reducing the sim-to-real gap in the training data. This is also supported by the quantitative gap between performance on real and simulated data. We have identified a number of issues in our training data, and propose strategies to resolve them:

1. Lack of sequences with periods of no motion - in the HQF frames, there are periods with little-to-no motion, and therefore no events. As shown in figure 4.16, our encoder does not retain the scene intensities in these situations. This is likely because these situations do not exist in the training data. To solve this, we can employ the same pause augmentation strategy as E2VID [3], and occasionally set the input events to zero, using the previous ground truth to compute the loss.

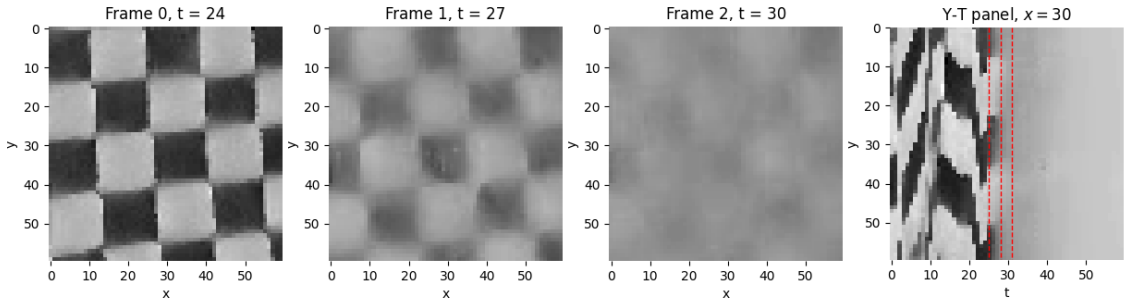


Figure 4.16: Example of failure when there are no input events, taken from the *slow_hand* scene in HQF. The timestamps of the three frames are shown by the red dashed lines in the Y-T panel. Lack of motion for only 6 event frames makes the encoder "forget" the scene.

2. Continuous optical flow - in our training data, we use continuous affine transformations applied to still images to generate scenes. However, this leads to a spatially continuous optical flow, and cannot for example model multiple objects moving independently against a still background. We believe that this unrealistic motion might introduce biases that widen the sim-to-real gap. To solve this, we can generate affine matrices for multiple still images, and then overlap them to generate the scene.

3. Contrast threshold - Stoffregen et al. have shown that the contrast threshold (CT) is a "key simulation parameter that impacts performance of supervised CNNs" [23]. In our method, we use a fixed CT of 0.35, as the simulator intrinsically introduces significant noise to this. We reasoned that this noise in combination with our stochastic pre-processing function would let the method generalize to real data. However, preliminary analysis showed that when running the HQF intensity frames through the simulator, the number of events we generated was roughly three times smaller than the real number of events. To achieve a similar number of events, we had to set the CT to 0.18. This suggests that employing a similar strategy to [23] and generating events for a wide number of CTs will likely help bridge the sim-to-real gap.

5

Conclusion

In this thesis, we presented a novel Local Implicit Function model for spatiotemporally continuous video representation from event data. We explored its application in video reconstruction, spatiotemporal superresolution, and uncertainty quantification.

On video reconstruction, we achieved performance similar to comparable state-of-the-art methods on real-world datasets. We identified the quality of simulated training data as a potential limitation and suggested several strategies to bridge the gap between simulated and real data.

We showed that for spatial superresolution, our method significantly outperforms a simple baseline up to 3x upscaling on the simulated data. For temporal superresolution, similar conclusions are difficult to draw due to limitations in the data, and we suggested a method to evaluate this more thoroughly. We also presented examples from real-world datasets, showing that our method generalizes to real event data.

We also proposed a novel metric for evaluating uncertainty predictions, and argued that it can be more informative than the usual metrics in the context of downstream computer vision applications. We analyzed how our predicted uncertainties correlate with perceptual reconstruction errors and suggested a simple heuristic to potentially enhance downstream vision algorithms. Despite a degradation in traditional uncertainty metrics when our method is applied to real data, the performance on our novel metric remained similar to simulated data, indicating that the method might be applicable to real-world scenarios.

Looking forward, we identified several avenues for continuing the work presented in this thesis. First, addressing the sim-to-real gap is essential, as it could substantially enhance the performance of our model. Benchmarking our spatial superresolution approach against state-of-the-art methods would help contextualize our performance. Additionally, a more in-depth analysis of temporal superresolution could show how well our method captures the high temporal resolution of event cameras. Lastly, further exploration into potential applications of our uncertainty predictions on downstream vision algorithms is needed.

Bibliography

[1] G. Gallego *et al.*, "Event-based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, Jan. 2022, arXiv:1904.08405 [cs], ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.3008413. [Online]. Available: http://arxiv.org/abs/1904.08405 (visited on 04/16/2024).

[2] D. Gehrig and D. Scaramuzza, "Are High-Resolution Event Cameras Really Needed?" en,

[3] H. Rebecq *et al.*, "High Speed and High Dynamic Range Video with an Event Camera," en, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 6, pp. 1964–1980, Jun. 2021, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2019.2963386. [Online]. Available: https://ieeexplore.ieee.org/document/8946715/ (visited on 04/16/2024).

[4] Y. Chen, S. Liu, and X. Wang, *Learning Continuous Image Representation with Local Implicit Image Function*, arXiv:2012.09161 [cs], Apr. 2021. [Online]. Available: http://arxiv.org/abs/2012.09161 (visited on 04/16/2024).

[5] Z. Chen *et al.*, *VideoINR: Learning Video Implicit Neural Representation for Continuous Space-Time Super-Resolution*, arXiv:2206.04647 [cs, eess], Jun. 2022. [Online]. Available: http://arxiv.org/abs/2206.04647 (visited on 04/25/2024).

[6] D. Nix and A. Weigend, "Estimating the mean and variance of the target probability distribution," en, in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Orlando, FL, USA: IEEE, 1994, 55–60 vol.1, ISBN: 978-0-7803-1901-1. DOI: 10.1109/ICNN.1994.374138. [Online]. Available: http://ieeexplore.ieee.org/document/374138/ (visited on 04/17/2024).

[7] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128$\times$128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," en, *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008, ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337. [Online]. Available: http://ieeexplore.ieee.org/document/4444573/ (visited on 05/19/2024).

[8] A. Glover and C. Bartolozzi, "Robust visual tracking with a freely-moving event camera," en, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC: IEEE, Sep. 2017, pp. 3769–3776, ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8206226. [Online].

Available: http://ieeexplore.ieee.org/document/8206226/ (visited on 05/19/2024).

[9] J. J. Hagenaars *et al.*, "Evolved Neuromorphic Control for High Speed Divergence-Based Landings of MAVs," en, *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6239–6246, Oct. 2020, ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.3012129. [Online]. Available: https://ieeexplore.ieee.org/document/9149674/ (visited on 05/19/2024).

[10] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," *Science Robotics*, vol. 5, no. 40, eaaz9712, Mar. 2020, Publisher: American Association for the Advancement of Science. DOI: 10.1126/scirobotics.aaz9712. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.aaz9712 (visited on 05/19/2024).

[11] T. Taunyazov *et al.*, "Event-Driven Visual-Tactile Sensing and Learning for Robots," en, in *Robotics: Science and Systems XVI*, Robotics: Science and Systems Foundation, Jul. 2020, ISBN: 978-0-9923747-6-1. DOI: 10.15607/RSS.2020.XVI.020. [Online]. Available: http://www.roboticsproceedings.org/rss16/p020.pdf (visited on 05/19/2024).

[12] F. Baghaei Naeini *et al.*, "A Novel Dynamic-Vision-Based Approach for Tactile Sensing Applications," en, *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 5, pp. 1881–1893, May 2020, ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2019.2919354. [Online]. Available: https://ieeexplore.ieee.org/document/8723387/ (visited on 05/19/2024).

[13] A. Vitale *et al.*, "Event-driven Vision and Control for UAVs on a Neuromorphic Chip," en, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China: IEEE, May 2021, pp. 103–109, ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9560881. [Online]. Available: https://ieeexplore.ieee.org/document/9560881/ (visited on 05/19/2024).

[14] R. S. Dimitrova *et al.*, "Towards Low-Latency High-Bandwidth Control of Quadrotors using Event Cameras," en, in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France: IEEE, May 2020, pp. 4294–4300, ISBN: 978-1-72817-395-5. DOI: 10.1109/ICRA40945.2020.9197530. [Online]. Available: https://ieeexplore.ieee.org/document/9197530/ (visited on 05/19/2024).

[15] V. Vishnevskiy *et al.*, *Optimal OnTheFly Feedback Control of Event Sensors*, en. (visited on 11/22/2023).

[16] V. Sitzmann, *Awesome Implicit Representations - A curated list of resources on implicit neural representations*. [Online]. Available: https://github.com/vsitzmann/awesome-implicit-representations.

[17] H. M. D. Kabir *et al.*, "Neural Network-Based Uncertainty Quantification: A Survey of Methodologies and Applications," *IEEE Access*, vol. PP, Jun. 2018. DOI: 10.1109/access.2018.2836917.

[18] C. Scheerlinck, N. Barnes, and R. Mahony, "Continuous-Time Intensity Estimation Using Event Cameras," en, in *Computer Vision – ACCV 2018*, C. Jawahar *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 308–324, ISBN: 978-3-030-20873-8. DOI: 10.1007/978-3-030-20873-8_20.

[19] C. Brandli, L. Muller, and T. Delbruck, "Real-time, high-speed video decompression using a frame- and event-based DAVIS sensor," en, in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Melbourne VIC, Australia: IEEE, Jun. 2014, pp. 686–689, ISBN: 978-1-4799-3432-4. DOI: 10.1109/ISCAS.2014.6865228. [Online]. Available: http://ieeexplore.ieee.org/document/6865228/ (visited on 05/20/2024).

[20] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous Optical Flow and Intensity Estimation from an Event Camera," en, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 884–892, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.102. [Online]. Available: http://ieeexplore.ieee.org/document/7780471/ (visited on 05/20/2024).

[21] C. Reinbacher, G. Graber, and T. Pock, *Real-Time Intensity-Image Reconstruction for Event Cameras Using Manifold Regularisation*, en, arXiv:1607.06283 [cs], Aug. 2016. [Online]. Available: http://arxiv.org/abs/1607.06283 (visited on 05/20/2024).

[22] C. Scheerlinck *et al.*, "Fast Image Reconstruction with an Event Camera," en, in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Village, CO, USA: IEEE, Mar. 2020, pp. 156–163, ISBN: 978-1-72816-553-0. DOI: 10.1109/WACV45572.2020.9093366. [Online]. Available: https://ieeexplore.ieee.org/document/9093366/ (visited on 04/25/2024).

[23] T. Stoffregen *et al.*, *Reducing the Sim-to-Real Gap for Event Cameras*, en, arXiv:2003.09078 [cs], Aug. 2020. [Online]. Available: http://arxiv.org/abs/2003.09078 (visited on 05/02/2024).

[24] L. Wang *et al.*, "Event-Based High Dynamic Range Image and Very High Frame Rate Video Generation Using Conditional Generative Adversarial Networks," en, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 10 073–10 082, ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.01032. [Online]. Available: https://ieeexplore.ieee.org/document/8954323/ (visited on 04/23/2024).

[25] B. Ercan *et al.*, "EVREAL: Towards a Comprehensive Benchmark and Analysis Suite for Event-based Video Reconstruction," en, in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, arXiv:2305.00434 [cs], Jun. 2023, pp. 3943–3952. DOI: 10.1109/CVPRW59228.2023.00410. [Online]. Available: http://arxiv.org/abs/2305.00434 (visited on 05/02/2024).

[26] E. Shechtman, Y. Caspi, and M. Irani, "Increasing Space-Time Resolution in Video," in *Computer Vision — ECCV 2002*, A. Heyden *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 753–768, ISBN: 978-3-540-47969-7.

[27] U. Mudenagudi, S. Banerjee, and P. K. Kalra, "Space-time super-resolution using graph-cut optimization," eng, *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 995–1008, May 2011, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2010.167.

[28] X. Xiang *et al.*, "Zooming Slow-Mo: Fast and Accurate One-Stage Space-Time Video Super-Resolution," English, ISSN: 1063-6919, 2020, pp. 3367–3376. DOI: 10.1109/CVPR42600.2020.00343.

[29] M. Haris, G. Shakhnarovich, and N. Ukita, "Space-Time-Aware Multi-Resolution Video Enhancement," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 2856–2865, ISBN: 978-1-72817-168-5. DOI: 10.1109/CVPR42600.2020.00293. [Online]. Available: https://ieeexplore.ieee.org/document/9156388/ (visited on 05/21/2024).

[30] E. Mueggler *et al.*, "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM," en, *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, Feb. 2017, arXiv:1610.08336 [cs], ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364917691115. [Online]. Available: http://arxiv.org/abs/1610.08336 (visited on 05/02/2024).

[31] H. Rebecq, D. Gehrig, and D. Scaramuzza, "ESIM: An Open Event Camera Simulator," en, 2018.

[32] T.-Y. Lin *et al.*, *Microsoft COCO: Common Objects in Context*, arXiv:1405.0312 [cs], Feb. 2015. [Online]. Available: http://arxiv.org/abs/1405.0312 (visited on 04/18/2024).

[33] P. Virtanen *et al.*, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, arXiv:1907.10121 [physics], ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: http://arxiv.org/abs/1907.10121 (visited on 04/17/2024).

[34] A. Paszke *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, arXiv:1912.01703 [cs, stat], Dec. 2019. [Online]. Available: http://arxiv.org/abs/1912.01703 (visited on 04/17/2024).

[35] N. Ballas *et al.*, *Delving Deeper into Convolutional Networks for Learning Video Representations*, arXiv:1511.06432 [cs], Mar. 2016. [Online]. Available: http://arxiv.org/abs/1511.06432 (visited on 04/25/2024).

[36] D. Ulyanov, A. Vedaldi, and V. Lempitsky, *Instance Normalization: The Missing Ingredient for Fast Stylization*, en, arXiv:1607.08022 [cs], Nov. 2017. [Online]. Available: http://arxiv.org/abs/1607.08022 (visited on 04/25/2024).

[37] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, arXiv:1511.07289 [cs] version: 5, Feb. 2016. [Online]. Available: http://arxiv.org/abs/1511.07289 (visited on 04/25/2024).

[38] J. Johnson, A. Alahi, and L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, en, arXiv:1603.08155 [cs], Mar. 2016. [Online]. Available: http://arxiv.org/abs/1603.08155 (visited on 04/25/2024).

[39] R. Zhang *et al.*, *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*, arXiv:1801.03924 [cs], Apr. 2018. [Online]. Available: http://arxiv.org/abs/1801.03924 (visited on 04/25/2024).

[40] L. Sluijterman, E. Cator, and T. Heskes, *Optimal Training of Mean Variance Estimation Neural Networks*, en, arXiv:2302.08875 [cs, stat], Aug. 2023. [Online]. Available: http://arxiv.org/abs/2302.08875 (visited on 04/25/2024).

[41] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, en, arXiv:1412.6980 [cs], Jan. 2017. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 05/01/2024).

[42] L. Sluijterman, E. Cator, and T. Heskes, *How to Evaluate Uncertainty Estimates in Machine Learning for Regression?* en, arXiv:2106.03395 [cs, stat], Aug. 2023. [Online]. Available: http://arxiv.org/abs/2106.03395 (visited on 05/02/2024).

[43] Hyndman, {Robin John} and George Athanasopoulos, "Prediction intervals," English, in *Forecasting: Principles and Practice*, Australia: OTexts, 2018. [Online]. Available: https://otexts.com/fpp2/prediction-intervals.html (visited on 05/03/2024).

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY