SAPIENZA
UNIVERSITÀ DI ROMA

Learning to Reconstruct Color Images from Event-based Neuromorphic Cameras using a Transformer Architecture

Faculty of Information Engineering, Informatics, and Statistics
Computer Science Master Degree

Fabrizio Rossi
ID number 1815023

Advisor
Prof. Luigi Cinque

Co-Advisors
Prof. Danilo Avola
Dr. Marco Cascio

Academic Year 2021/2022

Thesis defended on 19 January 2023
in front of a Board of Examiners composed by:

Chierichetti (chairman)
M.Mancini
Cilli
Cinque
Monti
Panizzi
Temperini

Learning to Reconstruct Color Images from Event-based Neuromorphic Cameras using a Transformer Architecture
Tesi di Laurea Magistrale. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: fabrizio.rss98@gmail.com

Dedicated to
My family which has always supported me in this journey,
My friends that were always there to cheer me up in hard times,
My advisors that were always available to help.

Contents

Chapter 1

Introduction

In this chapter, event-based vision and event cameras will be introduced, focusing in particular to their possible applications in Section 1.1 and reviewing the current state of the art in Section 1.2. The main contributions of this work will be presented in Section 1.3 where it will also be explained the structure of this entire document.

1.1 Event-based vision and Event Cameras

Nowadays Computer Vision is an ever evolving research field, producing new incredible results every week or so, allowing us to process visual data like we never did before. A multitude of sensors are used in this field to capture the environment, but the main ones are conventional cameras that produce both still images or smooth videos, by capturing a sequence of frames shot at a fixed rate.

However, depending on the end goal task that one wants to accomplish, this kind of sensors might not be the best choice, especially when deployed in challenging situations like capturing fast-motion scenes or when mounted on low-powered devices. This is why in the Computer Vision field, also other types of sensors are employed either in conjunction with normal cameras or entirely replacing them. Examples of additional sensors are LiDARs (Laser Imaging Detection and Ranging) [5], ToFs (Time of Flight) [22] or dToFs (direct ToF) sensors [56], while RGBD cameras [45], and stereo cameras [20] are some alternative examples of different types of cameras.

In this scenario, another kind of devices started to be commercialized in 2008, taking the name of Dynamic Vision Sensors (DVS) [25] and later often referred to as Event Cameras, requiring adaptations of traditional Computer Vision algorithms to a new paradigm.

This is because Event Cameras are neuromorphical devices biologically inspired to the human brain, that capture the environment around us in a profoundly different way from regular cameras.

In fact, instead of sampling light intensity at fixed intervals defined by a clock unrelated to the scene being acquired, they rely on the dynamics of the subject that is being captured by using their DVS sensors that independently and asynchronously react to brightness changes, enabling event-based vision. In particular, each pixel initially acquires the brightness intensity from the environment and stores its value (typically in a logarithmic scale). Then, it continuously keeps sampling the intensity

signal awaiting for changes, which are detected when the magnitude of the difference between the stored value and the one newly acquired exceeds a predefined value, called *contrast threshold*. When this happens, a piece of information called *event* is produced by the camera, containing the coordinates of the pixel that detected the change, the time when it happened and the so-called *polarity* which can be +1, when the brightness change is positive, or -1 when it's negative. This information is then transmitted externally from the camera using a shared digital output bus.

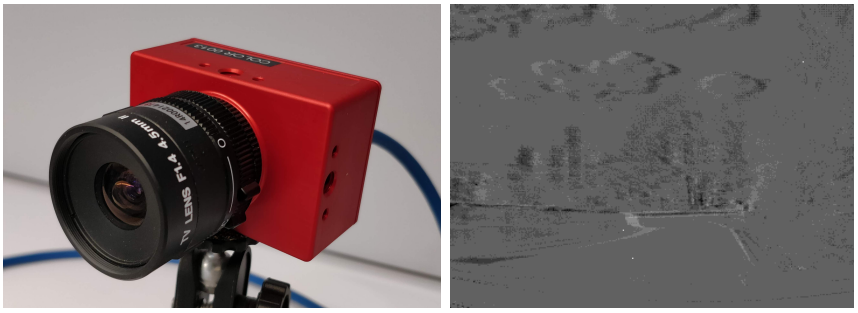

Figure 1.1. On the left a picture of the DAVIS346 event camera, while on the right an example of how these cameras capture the environment showing only brightness changes.

With respect to traditional cameras, Event Cameras offer some major advantages. First of all, events are not produced at a fixed rate and therefore Event Cameras are fully data-driven sensors: the more movement is present in the scene, the more data will be produced by the pixels that will promptly detect many brightness changes. Due to the asynchronous nature of the sensors, a brightness change can be detected and transmitted externally to another device in the order of microseconds, resulting in low latency transmission with a very high temporal resolution: existing commercial and academic devices have a sampling frequency ranging from 2 MHz to 1200 MHz. Moreover, they have a very high dynamic range, achieving 140 dB versus the 60 dB of regular cameras, making them optimal to capture scenes both in low-light and extremely bright conditions. Thanks to the absence of the exposure time, event cameras are also notorious to not suffer from the motion blur effect, which in regular cameras results in fast-moving objects leaving a streak over multiple frames, since the time needed to capture accurately the required amount of light might be too long to be captured in a single frame. Finally, they are also suited for low-powered devices since they transmit information only when the scene captured changes, removing redundant data: the typical system-level power consumption of such devices is 100 mW or less [44].

Due to the advantages mentioned before, Event Cameras have been firstly deployed in real-time interaction systems like robotics and wearable electronics, solving numerous tasks like object tracking [29, 1], Simultaneous Localization and Mapping (SLAM) [54, 21], optical flow estimation [3, 58, 14] and gesture recognition [52, 6].

However, even though many solid algorithms and models already exist to process images, they cannot be used with event data and this is why, since the commercialization and diffusion of Event Cameras, numerous works have proliferated trying to solve common Computer Vision tasks using directly event streams as input data,

showing good results [15, 24].

Despite this, other works instead went in the opposite direction, trying to reconstruct regular images, in order to be able to use already existing pre-trained models to solve downstream tasks [7, 41, 36]. This is partially why many different improvements of Event Cameras have been designed, trying to include more sensors in addition to the base DVS pixels that are only able to detect brightness changes. The two most notable examples are the Asynchronous Time-based Image Sensor (ATIS) [34] and the Dynamic and Active Pixel Vision Sensor (DAVIS) [4], that besides providing relative intensity changes (-1, +1), they also capture absolute intensity values in the form of grey-level values. To do so, in ATIS devices, each sensor is equipped with a DVS sensor and a conditional exposure measurement circuit. When the first one detects a brightness change, the second is used to capture an absolute measurement of the brightness intensity. In DAVIS devices instead, each DVS pixel shares the space with an additional Active Pixel Sensor (APS), already used in regular cameras to capture light. At a constant frame rate, all the APSs can be triggered simultaneously in order to generate normal images in the form of full intensity frames.

However, the images that can be acquired from these additional sensors are still prone to the defects of regular cameras, like low dynamic range and motion blur. This is why the topic of this thesis is to reconstruct an absolute intensity frame (or a sequence of them) leveraging only the event stream produced by an Event Camera, without using data coming from additional sensors. In order to do this, it's possible to inspect the events that have been produced, containing relative brightness changes, indicated by the polarity in the form of increments or decrements of brightness. Using this information, it might seem trivial to reconstruct the absolute value of intensity of a pixel at any time t, because it can be retrieved by computing the arithmetical sum between its initial intensity value and the different polarities recorded up until t for that pixel. However, this only works in ideal conditions without noise, with perfectly fixed contrast thresholds and if an initial full intensity frame is available, which is not the case when using only events data. Although image reconstruction is a difficult task, once an image frame is obtained, it is then possible to use any of the available Computer Vision algorithms out-of-the-box, benefiting from the high-temporal resolution and high dynamic range provided by Event Cameras.

1.2 State of the Art

In the state of the art there are numerous works that address the image reconstruction task, either by using exclusively events information or by fusing it with complementary information, such as absolute intensity frames acquired by APS sensors like the one mounted on the DAVIS cameras. In the following subsections, a review of the state of the art is presented, classifying different approaches according to the techniques used.

1.2.1 Hand-crafted methods

One of the first approach employed to reconstruct full intensity frames is to use filters, which are a typical tool used in signal processing in order to obtain predictable

results.

In [41] is investigated the possibility to use events complementing them with image frames in order to produce an high-temporal-resolution and high-dynamic-range image state. In this case the two sources of information are fused using a complementary filter which is particularly suited for scenarios where one signal has low-frequency disturbance and the other one has high-frequency disturbance. In fact, events provide reliable high-frequency data, while the acquired frames provide reliable low-frequency data. Their method consists in formulating the filter as an ordinary differential equation (ODE) and instead of using a sliding window approach, they update the image state in an asynchronous event-driven scheme that uses the solution to the ODE in order to update the generated intensity frame whenever a new event is generated. In this way they manage to produce a sequence of frames which is continuous in time and has very low latency.

Another approach to fuse these two kinds of information is presented in [53], where an Asynchronous Kalman Filter (AKF) is used to reconstruct HDR videos while reducing motion blur. In this case they perform a pre-processing step called frame augmentation in which they apply a de-blurring algorithm to the traditional full intensity frames. Then, the resulting images are temporal interpolated in order to match the events frequency and reduce ghosting problems. Successively, they compute the per-pixel gain for the filter using a proposed noise model, composed by three noise terms based on real camera hardware limitations. The augmented frames and event data are finally fused using an Asynchronous Kalman filter allowing the image state to be updated as soon as a new event is produced.

In [38], instead, they formulate the image reconstruction task as an energy minimization problem, modeling camera noise as multiple different data-terms and adding a regularization term in order to add smoothness to the solution. Also their method acts on a per-event basis, going along the asynchronous nature of events and not requiring events integration. In order to model the problem, a manifold induced by the most recent timestamp is created for each sample, typically called surface of active events. Then they formulate a variational model directly on the manifold and use the Primal-Dual algorithm to minimize it.

1.2.2 Spiking Neural Networks (SNNs)

In the last decade, thanks to the diffusion of publicly available datasets and more powerful hardware like GPUs and TPUs, Artificial Neural Networks (ANNs) have been revolutionizing the landscape of the state of the art in so many fields. Event cameras, being recently developed devices, are therefore not exempt from being processed with this kind of architectures. Moreover, since they are neuromorphic devices inspired by how the human brain transmits information among neurons, it was natural to use neural networks which fit more the asynchronous nature of events in order to process them. Thus, Spiking Neural Networks (SNNs) [27] have been used to process events, which mimic more closely the functioning of the brain while being more powerful, but most importantly more efficient, than regular neural networks.

There are a number of works that use SNNs to process event data, but they require event-based hardware which is not so wide-spread, and so there are only a

few that address the image reconstruction task.

In [59] they present for the first time a fully spiking neural network which utilizes Leaky-Integrate-and-Fire (LIF) neurons and a Membrane Potential (MP) neuron to address the image reconstruction task. This network, however, while requiring very low energy consumption, it is pretty limited in the results and that's why the authors also propose to augment the architecture with an additional potential-based model, containing an innovative adaptive membrane potential (AMP) neuron to enhance the temporal receptive field of the network.

Another work that uses SNNs is [10], in which they work in tandem with regular CNNs. In this case, first they preprocess the events in order to represent them as a tensor by summing their polarity pixel-wise. Then, they use a CNN in order to infer the Laplacian of the given frame tensors by using a combined loss. Subsequently, the trained CNN is converted into a SNN by using the NengoDL library. Finally, the predicted Laplacian is given as input to a feedforward SNN that implements Poisson integration. By using this architecture, they are able to obtain good results using as low as 200 parameters.

1.2.3 Convolutional Neural Networks (CNNs)

When processing image data, Convolutional Neural Networks (CNNs) [13] proved to be the go-to model thanks to their efficiency and data priors, so it's not a surprise that they are one of the main methods used also in the task of image reconstruction.

One of the first work to employ them is [33], in which they interpolate intensity frames using a fully-convolutional encoder-decoder architecture by starting from a single initial frame or by using a set of keyframes shot at a periodical rate. As first step, they process the events, creating what they call an event frame, which is simply the integration of the events polarity happened in a certain time interval, noting that there is loss of information when the sum of their polarity exceeds the 256 values threshold typical for images. Then, a starting frame and the event frame are given as input to the convolutional model that follows the famous UNet architecture [39], typically used in image segmentation, in order to infer the next frame. For the training procedure they simply compute the MSE between the reconstructed frame and the ground truth.

Successively, in [36], Rebecq et al. try to improve the previous model, arguably producing one of the most important work for this task, in which is also demonstrated how the reconstructed full intensity frames can be used for downstream tasks like object classification. In addition, since not many datasets are available, they generate synthetic training data and show how the model is able to generalize to real data. Concerning the method, initially the authors transform the event stream into a fixed-size 3D volume using a voxel grid in which events polarity is accumulated and distributed in temporal bins using trilinear interpolation. Then, along some previously produced frames, the voxel grid is processed through a Recurrent Neural Network (RNN) in which the main module is again inspired by the UNet, in order to reconstruct the next frame. In contrast to the previously mentioned work, as training loss they use the LPIPS [57] in order to obtain perceptually better results.

Additionally, in their follow-up paper [37], they make some improvements: first of all, as shown in the architecture in Figure 1.2, they don't rely anymore on

previously produced frames passing them directly to the model, but instead they use stacked ConvLSTMs in order to maintain an internal state, which can learn from an arbitrarily long sequence of past events. Moreover, they also include an additional loss, the temporal consistency loss, which enforces temporal consistency between successive reconstructions. Furthermore, they also show a method to leverage the reconstruction of grey scale images in order to output color images, starting from events produced by cameras using a Bayer filter in front of the sensor. In order to do that, they first reconstruct the single channels in grey scale at a quarter resolution, and then using a full resolution grey scale image, they combine everything in the LAB color space to obtain a final full resolution image in RGB.

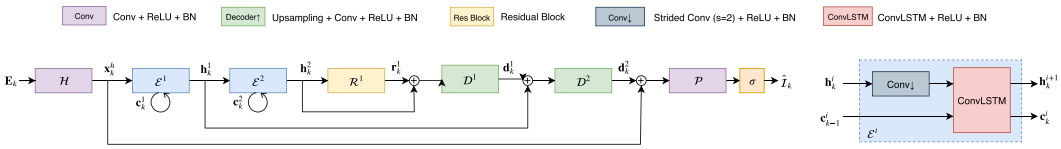

Figure 1.2. Architecture of the follow-up paper of E2VID [37].

However, this architecture requires lots of parameters and might not be suited for low-powered devices, which are one of the target of event cameras thanks to their ability to transmit only non-redundant data. This is what [42] tries to address, presenting FireNet, which requires less parameters, memory and computing power, running three times faster than the previous work. In this case they preprocess the events again in order to obtain a voxel grid, which this time is processed by a CNN used in combination with a Gated Recurrent Unit (GRU), in order to reduce the number of parameters.

Another work that uses CNNs is [18] in which they not only reconstruct an image starting from events, but they also produce a super-resolution version of it. In order to do that, first they create a representation of events by stacking a certain fixed number of them, multiple times. However, for each event they don't sum the polarity, but the corresponding pixels are initialized at 128 and their value is set to 256 if a positive event happened, or to 0 if a negative one happened. Then, 3 stacks of events are collected, one corresponding to an APS frame, one before and one after. The latter two are fed to the first module of the model, a pre-trained convolutional model which is going to estimate the optical flow of the events. The output is then concatenated with the events and rectified by a network composed by two convolutional layers that prevents blurry images. The result is successively processed by the main module, an RNN ispired by the ResNet architecture, which is going to perform super-resolution. Finally, a mixer network is used to increase the details richness. As training loss they use a combination of the L1-loss and the LPIPS. The authors also published a follow-up work [30] in which they improve temporal consistency by using an additional CNN called connection network and by adding a new term to the training loss. Finally in the paper they also show additional results on a colored events dataset using the same method reported for [37].

In any case, CNNs are typically used in combination with other architectures and techniques that are described in the following sections.

1.2.4 Generative Adversarial Networks (GANs)

Recently, one of the most popular architectures that are used for image synthesis are Generative Adversarial Networks (GANs) [16], since they have shown impressive results. The main idea behind them is to have two modules, a Generator that reconstructs the image frames and a Discriminator that instead checks the quality of the produced frames and tries to distinguish them from the ground truth frames. The training procedure is an adversarial game, where in turns the two modules compete having the generator outputting more and more convincing images and the discriminator getting better at not being fooled into classifying generated images as real ones.

Starting from this foundation, many other architectures have been proposed to improve the adversarial training. For example, in [32], a conditional GAN is used to perform the task of frame synthesis, which is the same as image reconstruction. Also in this case, they build an event frame by accumulating the polarity of events happened in a certain time interval to use it as input for the model which is composed by a Generative module and a Recurrent module. The first one infers the next frame, and the second one refines the output relying on the temporal coherence of the previously produced frames. Even in this case, the generator follows the UNet architecture, while the recurrent model is based on Convolutional LSTMs [47].

Another interesting work using GANs is [49], where not only is addressed the task of image reconstruction, but they also perform frames restoration and super-resolution. Their proposed pipeline is composed by three phases, that are of increasing difficulty. First, they reconstruct low resolution (LR) frames from events, which are typically blurred and noisy, then they restore them, and finally super-resolve the results into high quality images. Also in this case, they use simulated data but this time it's mixed with real data in order to form the training set. To preprocess the events, they create the usual 3D volume, accumulating a pre-defined number of events and summing their polarity. This representation is then given as input to the model that follows the previously mentioned phases, having a generator and a discriminator module for each of them. As training loss they use a linear combination of multiple terms: the adversarial loss typical of GANs, an event similarity loss used to reconstruct the original events from the produced image, an identity loss to avoid brightness and contrast variation among different iterations, and finally a total variation loss, to impose the spatial smoothness of the generated images.

The same authors also produced a follow-up work [50] in which they introduce some additions to the previous model. They increase the capacity of the model by adding non-local modules (NLMs) and they improve the training procedure in two ways: since they are training jointly multiple GANs, they use a stochastic noise perturbation approach which has been shown to stabilize the training, and they also use a Transfer Learning approach in order to perform the three phases. Interestingly, they also show results on color images, obtained again from events coming from a camera with a Bayer filter in front of the sensor. In this case they simply upscale each color channel from quarter resolution to full resolution and they run their grey scale model to obtain a full resolution color reconstruction.

1.2.5 Transformers

Not long ago, another architecture has been adapted to Computer Vision in order to process images. Transformers [48] in fact, are an architecture which has been extensively used in Natural Language Processing (NLP) for all kind of different tasks to elaborate text, but recently has been also converted to work with different type of data, including images [9] and, of course, events. However, for the task of image reconstruction, only a small number of works has been produced using Transformers, which are extremely good at processing sequences like events streams. In fact, at the time of writing, there are only two works that are here reported.

In [55], an hybrid architecture called ET-Net is presented to reconstruct videos from events, employing the strengths of CNNs for extracting local information and Transformers to build a global context. As is common for this task, also in this paper the events are first preprocessed in order to build a fixed size tensor, by summing the polarity of a certain number of events and distributing it into temporal bins using temporal bilinear interpolation. The result, as shown in Figure 1.3, is then given in input to a Recurrent Convolution Backbone (RCB), a recurrent convolutional network composed by three stacked blocks, that outputs features at three different scales. These features are then processed by a Token Pyramid Aggregation (TPA) module that using Transformers, models internal and intersected dependencies among the extracted features producing some hidden tokens. Finally, in order to output a tensor of the same size of the frame, a Multi-Level Upsampler (MLU) processes the hidden tokens and the feature pyramid to reconstruct a full intensity frame. As training loss, a linear combination of the LPIPS and a temporal consistency loss is used.

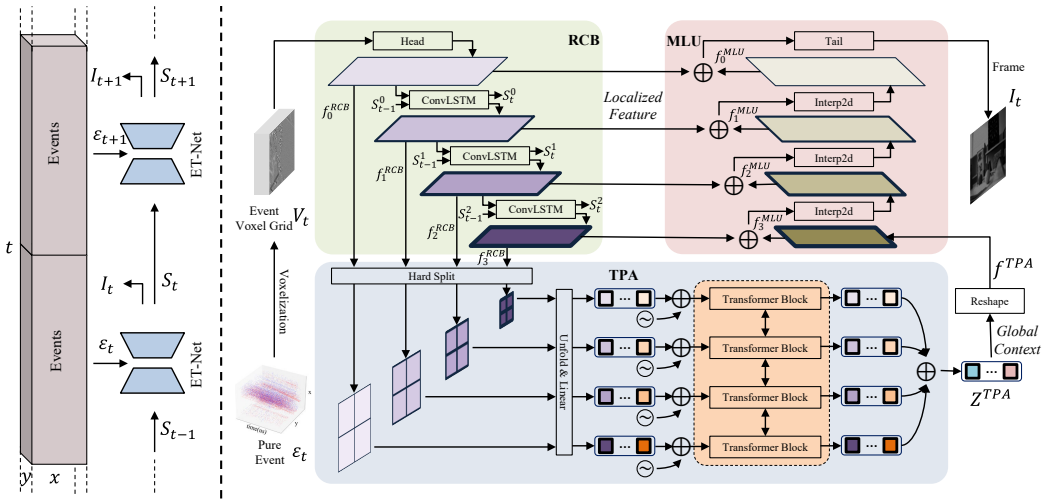

Figure 1.3. Architecture of the ET-Net model [55].

Instead, the other work is [46], that proposes SpikeNet, a model based on an encoder-decoder architecture using Transformers, and uses a simulator in order to generate training data, adding some noise to mimic real events. In this work they preprocess events differently from other papers: they consider spikes frames, which are binary frames coming from the continuous exchange of event information from

the camera, and divide them into patches which are then encoded by interlacing temporal and spatial information. The result is then given as input to four blocks of Transformer encoders that produce multi-scale features. Those are then merged and upsampled to the original image resolution by the decoder composed by MLP layers. The loss used is composed by four terms: the L1 and edge loss, that act locally, a perceptual loss that acts globally by computing the difference of the features extracted from the fifth layer of a VGG, and a temporal loss to introduce temporal constraints.

1.3 Contributions and Structure of the document

Currently there is quite a good number of datasets containing black and white events and images, but the same cannot be said for color events: at the time of writing of this thesis, the CED Dataset [43] is the only one publicly available, since the equipment required to capture sequences is expensive and collecting data is obviously a time consuming operation. This is why one of the contribution of this work is the creation of a synthetic dataset of color events and corresponding images by using the ESIM simulator [35].

In addition, all the models reviewed in the state of the art take as input black and white events and while some of them employ pre-processing techniques in order to produce color images (like treating each RGB channel as a single greyscale image), none of them uses directly color events. This work instead proposes EViT, a Transformer architecture that is able to process color events in order to reconstruct RGB images.

The rest of this document is composed by the following chapters:

- In **Chapter 2** is presented an in-depth overview on events, providing a formal definition and reviewing some processing techniques used to create different representations;
- In **Chapter 3** the concept of Deep Learning is introduced, discussing how models can learn from data, focusing on some particular architectures which have been used during this thesis;
- In **Chapter 4** is presented the EViT model, used in order to solve the task of image reconstruction starting from a color event stream;
- In **Chapter 5** implementation details are reported, discussing the creation and processing of the simulated dataset and showing the results obtained. At the end, also two other experimental architectures are reported;
- In **Chapter 6** some final conclusions are drawn, also suggesting some future works to improve the actual method.

Chapter 2

Events

This chapter contains an in-depth overview of how events are employed in the state of the art in order to solve numerous tasks. In particular, a formal definition of events in given in Section 2.1, different methodologies to process events are presented in Section 2.2 and finally some possible representations with advantages and disadvantages are discussed in Section 2.3.

2.1 Formal definition

Events are a small piece of information produced by Event Cameras when capturing the external environment. In particular, Event Cameras are composed by DVS sensors [25] that rapidly detect brightness changes by using photoreceptors. In order to do this, each pixel with coordinates (x, y), continuously samples the brightness intensity signal I and stores its logarithmic value at time t:

$$L(x, y, t) = \log(I) \tag{2.1}$$

A new event $e = (x, y, t, p)$ is produced when the difference ΔL between a new value acquired at time t and the one stored at $t - \Delta t$ exceeds a predefined limit C called *contrast threshold*, as expressed by Equation 2.3.

$$\Delta L(x, y, t) = L(x, y, t) - L(x, y, t - \Delta t) \tag{2.2}$$

$$|\Delta L(x, y, t)| > C \tag{2.3}$$

C can actually vary due to hardware limitations and in practice there can be two thresholds, one for positive changes C^+, and one for negative changes C^-. Moreover it is an important parameter that determines how many events are going to be produced by the camera: setting it too low means that tons of events are going to be generated in a short amount of time; while on the contrary, setting it too high means that little to none events are going to be detected.

The last component of an event is the *polarity* $p \in \{-1, 1\}$ which indicates the sign of the brightness change and can be formulated as:

$$\Delta L(x, y, t) = pC \tag{2.4}$$

It is interesting to note that events give us information about the temporal derivative. In order to prove that, it's possible to use the first order Taylor expansion to show that for a small Δt:

$$\Delta L(x, y, t) = L(x, y, t) - L(x, y, t - \Delta t) \approx \frac{\partial L(x, y, t)}{\partial t} \Delta t \tag{2.5}$$

and by rearranging the approximation:

$$\frac{\partial L(x, y, t)}{\partial t} \approx \frac{\Delta L(x, y, t)}{\Delta t} = \frac{pC}{\Delta t} \tag{2.6}$$

From here, assuming constant illumination and constant camera motion with velocity vector $\mathbf{v}$, it's also possible to show that events are generated by moving edges. This can be done starting from the Brightness Constancy Equation:

$$\frac{\partial L(x, y, t)}{\partial t} + \nabla_{x,y} L(x, y, t) \cdot \mathbf{v} = 0 \tag{2.7}$$

and using Equation 2.6:

$$\Delta L(x, y, t) \approx -\nabla_{x,y} L(x, y, t) \cdot \mathbf{v} \Delta t \tag{2.8}$$

which means that the brightness change ΔL is caused by a brightness gradient ∇L moving with velocity $\mathbf{v}$ on the image plane.

2.2 Processing

Since events represent a different kind of data with respect to natural images, there is also the need to employ techniques of event processing in order to use them in algorithms. In particular, events are a form of sparse information since each pixel operates asynchronously, but they have a very high temporal resolution. However, processing only the last events received does not provide sufficient information in order to get a complete picture of what scene is being acquired by the camera. This is why in the state of the art, there are two contrasting approaches that are employed when processing an event stream:

- **Event-by-event processing**: the system typically maintains an internal state where the events are stored, and as soon a new one is received, the state is updated to reflect the new information. The majority of algorithms that use this technique are deterministic and probabilistic filters which are usually used in conjunction with grey scale images that act as internal state, updated when a new event is acquired [41, 53]. However, there are also some examples of multi-layer Artificial Neural Networks that act on an event-by-event basis, especially leveraging Spiking Neural Networks, as a consequence of their matching impulse nature [40, 11]. Thanks to the promptly reaction of these algorithms, this type of processing is particularly suited for applications where efficiency and low-latency are required;

- **Event grouping processing**: when a new event is received, it is stored in a buffer, which is going to be processed by the algorithm only when a certain condition is met, for example when a predefined number of events have been collected (Stacking By Number - SBN), or when a certain period of time elapsed (Stacking By Time - SBT, typically in the range of seconds). Also other adaptive techniques can be implemented, like shown in [26] where they use as criterion the number of events in regions of the image plane. One advantage of event grouping processing is that it is robust to noise, since multiple events can provide a sufficient signal-to-noise ratio, but its disadvantage comes from increased latency.

2.3 Representations

Typically, events are not leveraged out-of-the-box, but instead they are transformed into a more suitable representation that can highlight particular features important for the algorithm. In the next list, a collection of different representation is reported:

- **Individual events**: an event $e = (x, y, t, p)$ is processed as it is, without further processing, expressing the coordinates of the pixel that triggered the event, its timestamp and its polarity. Deterministic and probabilistic filters are typical examples of algorithms that use this representation, joined by SNNs. Usually it's needed to use additional information derived from past observations in order to make sense of new events;

- **Event packet**: a certain number of events N_e in a spatio-temporal neighborhood are processed together: $\mathcal{E} = \{e_k\}_{k=1}^{N_e}$. In this case the events are not further processed like in the representations which will be described next, and therefore they contain the original information captured by the event camera. The choice of N_e is very important since it can affect the performance of the processing algorithm;

- **Event Frame or 2D Histogram**: events in a spatio-temporal neighborhood are converted into a 2D grid which is often represented as an image, grouping them by their coordinates and by counting their number or summing their polarity. Using this representation, spatial information is retained since the result is a map indicating where edges are located, while temporal information is discarded. This representation is a simple way to enable the use of already existing image processing algorithms;

Figure 2.1. Visualization of an event frame showing how edges information is preserved.

- **Time Surface (TS)**: a 2D grid in which each cell contains a single timestamp value, which is usually the one of the last event received by the pixel at that coordinates. They retain temporal information and can be easily updated asynchronously, but they are not effective when pixel values change frequently since old values are rapidly overwritten;

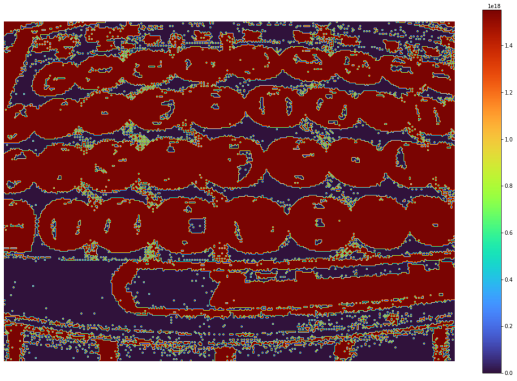

Figure 2.2. Visualization of a time surface showing which pixels are more active.

- **Voxel Grid**: events are stacked in a 3D space-time histogram of events, where each voxel represents a certain state of a pixel in a specific point in time. Usually a voxel contains information about the polarity of events, which is accumulated in a single voxel or spread among their closest voxels in the temporal dimension. In this case, even if time is quantized, the representation offers an effective way to express both temporal and spatial information. Thanks to its fixed size, it is mainly used for Deep Neural Networks like CNNs in order to extract features, and this is also why this representation is employed in the proposed method;

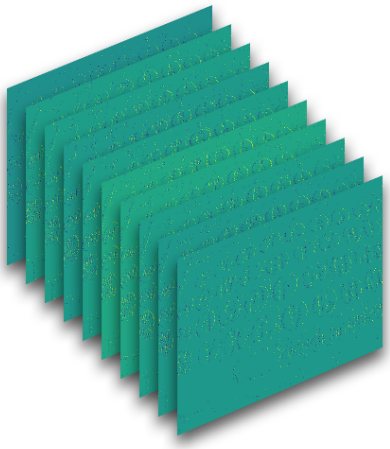

Figure 2.3. Visualization of a voxel grid interpreted as a stack of 2D temporal bins.

- **3D point set**: each event is represented by a point in a 3D space with its coordinates and timestamp. This is typically used in point-based geometric processing methods;

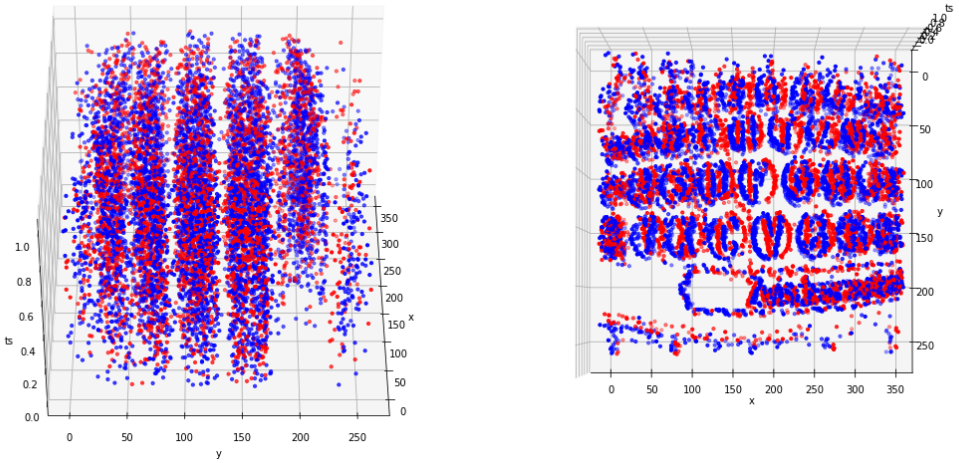

Figure 2.4. Visualization of a 3D event points set from two different angles. Red points have positive polarity, while blue points have negative polarity.

Chapter 3

Deep Learning

In this chapter the Deep Learning research field is introduced by giving some initial definitions of the Machine Learning research field in Section 3.1, and discussing the main categories of techniques in Section 3.2. Successively, the structure of Artificial Neural Networks (ANNs) is presented in Section 3.3 and the needed learning algorithm is described in Section 3.4. Finally, in Sections 3.5 and 3.6, Convolutional Neural Networks (CNNs) and Transformers, two particular neural networks architectures used in this thesis, are reviewed.

3.1 Introduction to Machine Learning

Machine Learning is a subset of the Artificial Intelligence research field that tries to emulate the human brain in order to automatically learn how to solve a certain problem. The main innovation of Machine Learning is that developers don't need anymore to formulate algorithms in terms of finite steps in order to solve a problem, but algorithms directly learn from input data.

In fact, in the typical imperative programming paradigm, a problem is usually solved by explicitly turning into code some rules extracted from the developer's experience and/or domain knowledge. Instead, with Machine Learning, it's the algorithm itself that learns the rules, just by observing the input data and trying to minimize the errors committed in its answers. In fact, the performances of the algorithm can be measured according to a certain criterion which can be specified and varies from task to task.

The popularity recently gained by Machine Learned in the last decades is due to the possibility to develop algorithms to solve those problems for which describing rules is much harder than making the computer learn an algorithm itself. An example of them which is particularly difficult to solve by formulating rules, and therefore where Machine Learning instead excels, is object recognition in images: it is indeed achievable to describe features to classify certain objects, but it's much more effective to develop machine learning algorithms that can learn them.

The objective of a Machine Learning algorithm is to build a so-called *model* which, analyzing a certain number of instances of the problem, can acquire knowledge about how to solve that particular task, being able also to produce correct answers on instances it has never seen (*generalization capability*).

3.2 Machine Learning Techniques

In the Machine Learning research field, there are three main different techniques that differ in how the input data is composed:

- **Supervised Learning:** the available data is composed by a *training set* consisting in pairs (x, y) where x is input data and y is called *ground truth* data, representing the correct output that the model should produce when given as input x. The ground truth data is also typically referred to as *labels*. The learning algorithm consists in using the training set in order to optimize some internal parameters of the model that influence its output, minimizing a so-called *loss function* that expresses the error produced by the model;
- **Unsupervised Learning:** the algorithm is not provided with correct answers (labels) but the objective is to group the input samples according to some common features which are learned by maximizing the internal similarity of samples belonging to a same group;
- **Reinforced Learning:** this technique is usually employed in robotics, where an agent is located in a certain environment. The algorithm is going to learn a sequence of actions that actuate on the environment, in order to reach a certain goal. To learn the sequence, each action is associated with a *reward* that can be positive when a good action is executed, and negative when the agent performs a bad action.

3.3 Neural Networks

Neural Networks (NNs) are one example of Machine Learning algorithms proposed for the first time in 1943 [28]. They vaguely recall the cerebral structure typical of animals and in particular they try to mimic the operations that the brain executes in order to recognize relations connecting a multitude of data. This is why Neural Networks are based on units called *neurons*, that are connected among each other by links called *synapses*, in analogy to the human brain. The majority of modern networks are based on multiple levels, as can be seen in Figure 3.1, and data advances from one neuron to the next without the possibility to go back: this is why they are also called *feed-forward models*.

A particular model of Neural Networks is called the *MultiLayer Perceptron* which consists in multiple layers of neurons connected among each other. In particular, neurons in the first layer are called *input neurons* since they are used in order to input data into the model. The central layers, instead, are called *hidden neurons* since their outputs are not visible from outside and their job is to process the signal received as input from the previous layer and to send the result to the neurons in the next one. Neurons in the final layer are called *output neurons* since they compute the last result which can then be used from outside the model.

Each neuron is a computational unit and can be formalized as a non linear function that transforms input data $(x_1, x_2, ..., x_n)$ into an output y. Neurons from one layer are connected to each neuron of the next one through links called *weights*.

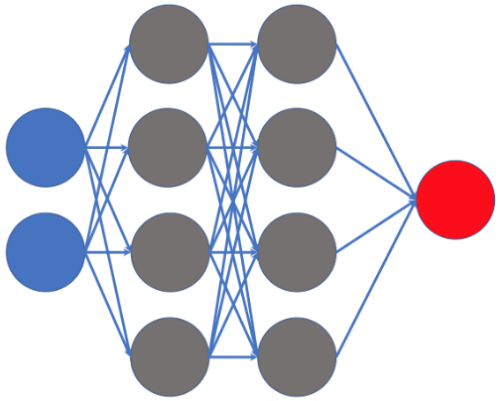

Figure 3.1. The structure of a MultiLayer Perceptron. Blue units are input neurons, grey units are hidden neurons, red units are output neurons.

The output of neuron is obtained starting by computing the weighted sum of the input values with respect to their corresponding weights, and usually another term called **bias** b is also added in order to be able to shift the output function:

$$a = \sum_{i=0}^{n} w_i x_i + b \tag{3.1}$$

The final output, indicated with y, is then computed by applying a function f which introduces non-linearity inside the network and determines if the current neuron is going to be activated:

$$y = f(a) \tag{3.2}$$

This function is therefore called *activation function* and must be differentiable due to the learning algorithm which requires the computation of gradients (see Section 3.4). During the years many different activation functions have been proposed in order to obtain different results and here are reported the most used ones:

- **Linear function**:
$$f(x) = x \tag{3.3}$$
represents the identify function, allowing the output to be the same as the input;

- **Step function**:
$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \tag{3.4}$$
can be used only in neural networks with a single layer since it's not differentiable. It is typically used for binary classification where data is linearly separable;

- **Sigmoid function**:
$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

since it's differentiable, it can be used for non-linear data and its values are in the range $[0, 1]$;

- **Hyperbolic tangent function**:

$$f(x) = \tanh x = \frac{\sinh(x)}{\cosh(x)} \tag{3.6}$$

it has output values in the range $[-1, 1]$ and in the learning phase enables the model to converge to correct solutions faster than the sigmoid function;

- **Rectified Linear Unit (ReLU)**:

$$f(x) = \max(0, x) \tag{3.7}$$

it has output values in the range $[0, \infty)$ and it's one of the most used functions in neural networks since it's very fast to compute, speeding up the learning procedure;

- **Leaky ReLU**:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{if } x < 0 \end{cases} \tag{3.8}$$

it's similar to the ReLU function, but it enables to have low intensity gradients when the neuron is not active.

3.4 Learning Algorithm

When developing a machine learning model, typically there are two phases: the *training* phase, in which an algorithm is used to find the optimal *parameters*, and an *inference* phase, in which the parameters are fixed and the model is used to predict answers for never seen instances. For the MultiLayer Perceptron, in the training phase, the learning algorithm enables to find the best weights values that minimize the so called *loss function*, which defines the task of the network and measures how incorrect the answers produced by the model are. In the case of supervised learning, one of the most used loss function is given by the Mean Squared Error (MSE) between the output of the network and the label assigned to each instance in the training set X, composed by n instances. In particular, denoting with W the set of weights assigned to neurons connections and $h(x)$ the result of the neural network for a certain training pair (x, y), the loss function can be defined as:

$$Loss(W) = \frac{1}{2n} \sum_{x \in X} (h(x) - y)^2 \tag{3.9}$$

In order to find the weights W that minimize this function, the *gradient descent* can be used: it is an optimization algorithm that enables to find one local minimum of a function. In particular, by computing the partial derivative of the loss function with respect to the weights, it's possible to analyze the characteristics of the gradient, defined in Equation 3.10.

$$\nabla Loss(W) = (\frac{\partial Loss}{\partial w_0}, \frac{\partial Loss}{\partial w_1}, ..., \frac{\partial Loss}{\partial w_n}) \tag{3.10}$$

In fact, the intensity and direction of the gradient indicate how much the loss function is increasing at a certain point, so in order to find a local minimum, it's sufficient to go towards the opposite direction. Therefore, weights can be updated iteratively using the *gradient descent rule* which is the following:

$$w_{ij} := w_{ij} - \eta \frac{\partial Loss}{\partial w_{ij}} \tag{3.11}$$

In the previous formula, w_{ij} represents a weight that connects a neuron i to a neuron j, and η is a multiplicative factor that determines how fast a neuron can learn, assuming therefore the name of *learning rate*.

Since the MultiLayer Perceptron is composed of multiple layers, it's necessary to update the weights according to their influence on the loss function. In order to do so, a technique called *backpropagation* is used, allowing the gradients to flow from the output going backwards.

To use backpropagation it's easier to focus on a single instance of the training set, and so the loss function becomes:

$$Loss_0(W) = \frac{1}{2}(h(x) - y)^2 \tag{3.12}$$

At this point, by indicating with a_j the output of a neuron j before the activation function, and with o_j the result of the activation function applied to a_j, it is possible to use the chain rule on the partial derivative of the loss function with respect to a weight w_{ij}:

$$\frac{\partial Loss_0}{\partial w_{ij}} = \frac{\partial Loss_0}{\partial o_j} \frac{\partial o_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \tag{3.13}$$

Since the formula changes whether a neuron is an output or hidden neuron, it's convenient to define $\delta_j = \frac{\partial Loss_0}{\partial o_j} \frac{\partial o_j}{\partial a_j}$ and so the previous equation becomes:

$$\frac{\partial Loss_0}{\partial w_{ij}} = \delta_j o_i \tag{3.14}$$

Now, for a neuron which is in the output layer:

$$\delta_j = (o_j - y) o_j (1 - o_j) \tag{3.15}$$

Instead, for a neuron in a hidden layer j, by calling K the set of neurons in the next layer:

$$\delta_j = (\sum_{k \in K} \delta_k w_{jk}) o_j (1 - o_j) \tag{3.16}$$

Finally, the entire learning algorithm is reported below:

Algorithm 1: Learning algorithm

Result: Trained model
Initialize weights with random values
while *the loss function error doesn't satisfy a certain criterion*
  for each *instance* (x, y) *in the training set* X
    compute the result of the network for x
    compute the error between the result of the network and the label y
    update the network weights using backpropagation
  end
end

Every time the algorithm analyzes all the instances contained in the training set, and updates the weights accordingly, it is said that it has completed an *epoch*.

Usually, the training set is not the only dataset used, but a part of the dataset is reserved to be used as a *test set*, containing instances that will not be used during the training procedure. For this reason, it is usual to track the loss obtained on the training and on the test set since it can give insights on how good the model can generalize on unseen instances.

By plotting these two values over the epochs, it is possible to notice that the model could possibly assume two bad behaviors which are called *overfitting* and *underfitting*, shown in Figure 3.2. The first one happens when the model fits perfectly the instances of the training set but produces instead bad results on the ones in the test set, making the model not generalize well. The second one instead is the opposite situation, when the model can't even obtain good performances on the training set, and it is usually a symptom that the architecture chosen for the model is not suited for the data or for the task that needs to be solved.

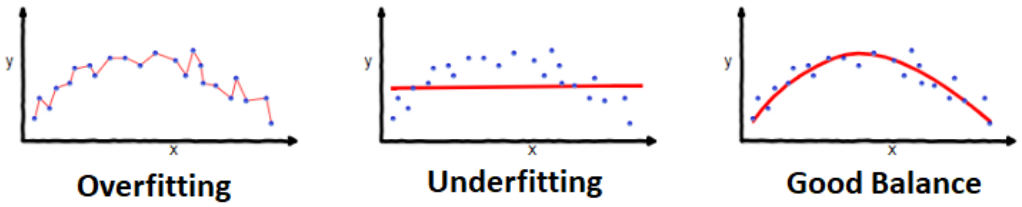

Figure 3.2. An example of overfitting, underfitting and good balance on a set of points. The red line represents the model output.

3.5 Convolutional Neural Networks (CNNs)

Over the recent years, lots of different variations of neural networks have been proposed and with the improvements brought to hardware, especially GPUs, networks with lots of hidden layers have been successfully trained showing impressive results, giving birth to the Deep Learning research sub-field which studies these incredibly deep architectures.

One of the most successful architectures in Deep Learning are Convolutional Neural Networks (CNNs) [13] which have achieved remarkable performances on

image processing tasks, like image classification [23]. This type of networks are particularly suited for images because they dramatically reduce the number of weights needed to process them and it has been shown that through their layers they are able to extract higher and higher level features, which can be successively elaborated in order to solve a specific task.

Their structure is inspired to the visual cortex of animals where neurons respond to stimuli relative to a restricted area, called *receptive field*. Usually multiple overlapping receptive fields are used in order to scan the entire view area. With respect to the MultiLayer Perceptron, a CNN preserves the structure of images since it doesn't analyze each single piece of information (pixel), but it focuses on a region of the image at a time, leveraging all the spatial information.

In order to do that, the main computation performed by CNNs is the *convolution* applied all over the image. The convolution operator $\otimes$ is a matrix operator, so it is need to interpret images as such. In fact, an image $I(x, y)$ can be formalized as a function that returns the intensity of the pixel at coordinates (x, y). The first operand of the convolution w is usually called *kernel* or *filter* and the operator is defined as:

$$G[x, y] = w \otimes I = \sum_{dx=-\infty}^{\infty} \sum_{dy=-\infty}^{\infty} w[dx, dy] I[x - dx, y - dy] \tag{3.17}$$

The output of the convolution is called *feature map* since it can be interpreted as a map that indicates how much a feature expressed by the kernel is present in the different parts of the image.

The convolutional layers that compose CNNs are based on this operation, in which kernels are parameters that can be learned in order to extract the most significant features from images. This means that there is no need to have a neuron for each pixel, but instead the weights of the kernels are shared for the whole image which is processed using a sliding-window approach. It is also relevant to note that the detection of image features using the convolution operator is location invariant.

Another layer which is very common in CNNs is the Pooling layer which behaves similarly to downsampling, reducing the output size. In this case a kernel is used as a sliding window all over the image but this time the pixels analyzed are summarized using a reducing operation which can be the max, min, sum or average. In CNNs typically MaxPooling layers are used since they are able to make the network more robust to small variations in the position of the features in the input image.

The ReLU activation function, introduced in Section 3.3, is very common in this type of networks, since it filters out negative activations and it's much faster to compute that the tanh or sigmoid activation functions, achieving similar generalization accuracy while still introducing non-linearity in the network.

An additional layer which is often used is the Batch Normalization layer [19] that can speed up training, allowing to use higher learning rates. This layer applies normalization not on the input data, but on the output of each layer, reducing the change of the data distribution inside the network (the so called *internal covariate shift*). To perform the normalization, the layer has two learnable parameters γ, β which are used to transform the output x of a layer according to the following formula:

$$\hat{x} = \gamma \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}} + \beta \tag{3.18}$$

Finally, there is one last trick that is typically used for training CNNs and in general also other architectures like Transformers: *residual connections*. They first appeared in [17] where a very deep CNN composed by up to 152 layers is trained successfully thanks to this mechanism. A residual connection is also called *skip connection* because it enables the data to flow not only to the next layer, but it adds another path which can skip a bunch of them. The definition is very simple since it computes the addition between the output x of a certain layer, with the output of one or more following layers which can be composed in a function F:

$$\hat{x} = x + F(X) \tag{3.19}$$

The effects of this mechanism are not well understood yet but empirically it has been shown that it helps models to converge much more easily, even when the architecture is very deep. This can be partially due to the fact that gradients can flow quicker through the first layers of the network using the new connections, alleviating the vanishing gradient problem.

3.6 Transformers

Another architecture that has been gaining a lot of traction lately are Transformers [48], which have shown great potential for sequence-to-sequence tasks. The original paper focuses on the Neural Language Processing (NLP) research field and in particular in machine translation, but since then lots of works adapted their architecture to many different kind of tasks and input data including images, showing promising results for example in image classification [9].

Transformers are not the first model which has been used for sequences, since Recurrent Neural Networks (RNNs) like Long-Short Term Memory (LSTMs) and Gated Recurrent Units (GRUs) have been used extensively, but they all have one problem in common: they have a finite memory that can't keep track of sequences which are too long. This is instead where Transfomers shine, since by using their Attention mechanism, they can actually focus on sequences of infinite length.

The original architecture of a Transfomer is shown in Figure 3.3 and is composed by multiple modules enclosed in a stacked Encoder-Decoder architecture. As said previously, one of the central modules is the Multi-Head Attention which takes as input three vectors of dimension d_k, called respectively *query* Q, *key* K and *value* V. The output is a weighted sum of the components of the *value* vector where the weights are assigned accordingly to the dot product between the query and key vectors. In particular, the weights are first scaled to prevent small gradients, and then a final softmax function is applied to obtain the final result:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3.20}$$

However, instead of using a single attention layer, in the paper they linearly project the three input vectors h times using learned weights and compute in parallel the attention function. The output vectors are then concatenated and projected again in the following way:

$$\begin{aligned} \text{MultiHeadAttention}(Q, K, V) &= \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.21}$$

In this case QW_i^Q, KW_i^K, VW_i^V are matrices which perform the projection on the input vectors and W^O is used to finally project the result.

This MultiHeadAttention is used in the encoder to learn relationships among the tokens in the input data and is called *self-attention* because the query, key and value vectors come from the same place which is the input tokens or the output of the previous layer. In the decoder instead, in addition to an initial self-attention layer, there is an additional MultiHeadAttention layer, which this time relates the output of the encoder with the input of the decoder by having as Q the output of the previous decoder layer and as K and V the output of the encoder.

Another important component of the architecture is the positional encoding, which enables to inject positional information in the model, since the attention mechanism can be computed in parallel, speeding up the training, but it's not aware of the original sequence of the input. This is done by adding positional encoding

vectors to the input embeddings of the encoder and decoder. In the sine and cosine schema proposed in the paper, noting with *pos* the position in the sequence of an input vector, and with i the index of a dimension, positional encoding vectors are computed as follows:

$$\begin{aligned} \mathrm{PE}(pos, 2i) &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ \mathrm{PE}(pos, 2i+1) &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned} \tag{3.22}$$

Both the encoder and decoder module have then some feed forward networks with skip connections and layer normalizations that can transform the embeddings created enriching them with information useful for the task.

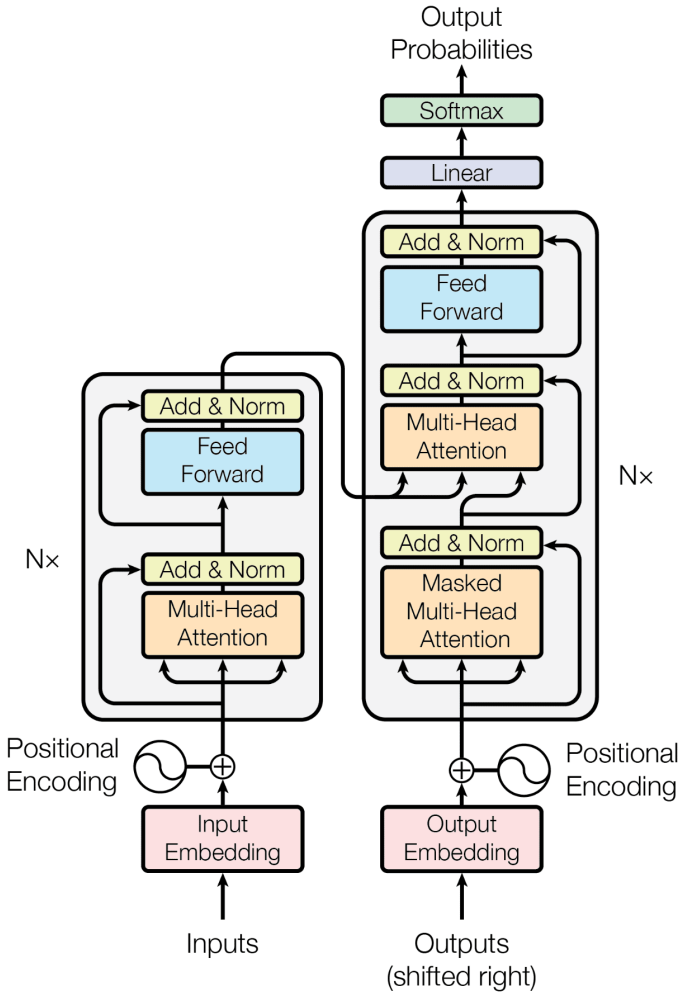

Figure 3.3. Architecture of a Transformer model taken from the original paper [48].

3.6.1 Vision Transformers (ViTs)

As previously mentioned, Transformers have been originally used in NLP but their Attention mechanism has proved to be successful in many other applications. This is the direction taken by Vision Transformers (ViTs) [9] that try to adapt the Transformer architecture to the image classification task, following as close as possible the original paper. CNNs have been the most successful architecture for image processing, but those models incorporate data priors like translation invariance and feature locality, while Transformers can learn any solution through the Attention mechanism, including functions similar to CNNs to extract visual features, and thanks to their ability to attend the whole input, they can also learn long range dependencies.

The architecture of ViTs is shown in Figure 3.4 and it can be seen that in the case of image classification, only the Transformer encoder is used, discarding the decoder. Images are not sequence data, so it's needed to process them into a suitable representation in order to be used as input tokens by the encoder. In the paper they propose to split the image into equally sized patches and to project them into 1D embeddings. Before being processed by the encoder, the vectors are summed with learnable positional encoding vectors to inject positional information.

In order to perform the classification, an additional learnable embedding `[class]` is added to the input sequence like in the BERT model, and its output produced by the Transformer encoder is used in order to perform the final classification by attaching an MLP head to it.

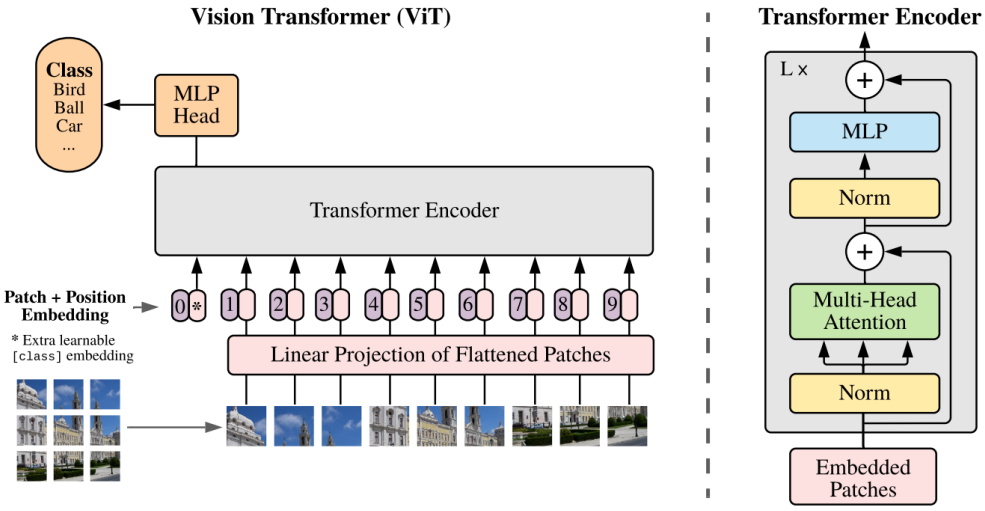

Figure 3.4. Vision Transformers architecture taken from the original paper [9].

Chapter 4

Method

In this chapter is presented the EViT model, proposed to perform the color image reconstruction task. In particular, in Section 4.1 is described the preprocessing needed to transform events into a fixed size representation, while in Section 4.2 the architecture of the model is explained in details. Finally, in Section 4.3 is described the loss that has been used in the training procedure.

4.1 Data preprocessing

The task of image reconstruction starts with data captured from Event Cameras which comes in the form of an event stream $\mathcal{E} = \{e_k\}_{k=1}^{\infty}$ where $e_k = (x_k, y_k, t_k, p_k)$, as described in Section 2.1. However, this format is usually not used directly in Deep Learning since the majority of models take as input a fixed-size tensor, and this is why, as described in Section 2.3, many works in the state of the art, including this one, employ the 3D voxel grid representation.

In order to obtain a fixed-size 3D volume $\mathrm{E} \in \mathbb{R}^{T \times H \times W}$, a certain number of events N_e are accumulated creating a finite set of events $\mathcal{E}_i = \{e_k\}_{k=1}^{N_e}$. The time spanned by those events is evenly divided into T temporal bins and their polarity is then summed and distributed into the two closest temporal bins according to the following formula:

$$\mathrm{E}[t, y, x] = \sum_{\substack{e_k \in \mathcal{E}_i \\ e_k = (x_k, y_k, p_k, t_k)}} p_k \max(0, 1 - |t - t^*|) \tag{4.1}$$

In this case, t^* is the bin to which the event should be assigned according to its normalized timestamp, retaining also the decimal part. In detail, noting with t_0 the timestamp of the first event and with t_{N_e} the timestamp of the last event in the sequence considered $\mathcal{E}_i$:

$$t^* = \frac{t_k - t_0}{t_{N_e} - t_0}(T - 1) \tag{4.2}$$

4.2 Event ViT (EViT)

The model proposed is called Event ViT (EViT) since it takes inspiration from the Vision Transformer (ViT) model [9]. At high level, its architecture is pretty straightforward and, as shown in Figure 4.1, it is composed of three main modules: a convolutional encoder followed by a Transformer encoder and a final convolutional decoder.

The first convolutional encoder is used to encode temporal and spatial information extracted locally from the input events. The feature maps obtained are divided into patches of fixed size in a similar fashion to the ViT model, and they are unrolled across the filters dimension. The patches are then flattened and summed with positional encoding vectors in order to form a sequence that is given as input to the Transformer encoder, which can correlate global dependencies among the local features. After recomposing back the feature maps, a convolutional decoder is finally employed in order to transform the enriched global features into an RGB image depicting the scene captured by the events processed.

The low-level details of the architecture of each module are described in the following subsections.

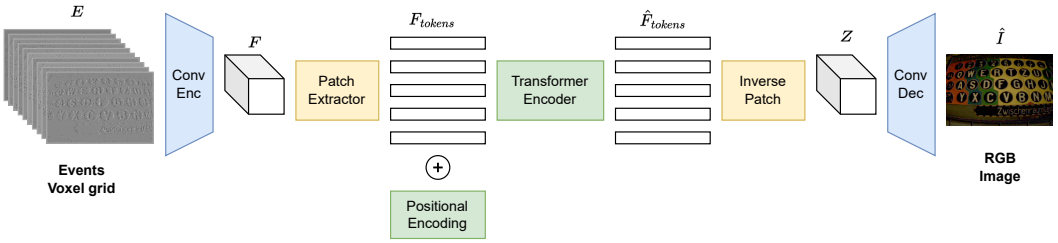

Figure 4.1. High-level architecture of the proposed model.

4.2.1 Convolutional Encoder

The first module of the architecture is a convolutional encoder that takes as input a voxel grid $E \in \mathbb{R}^{T \times H \times W}$ that has T temporal bins and a sensor size of $W \times H$. Its architecture is shown in Figure 4.2 and it is composed of 4 blocks, containing each a 2D convolutional layer, 2D batch normalization and the ReLU activation function. After the second block, a MaxPooling layer is employed in order to decrease the dimensionality of the output features maps. The result is a tensor $F \in \mathbb{R}^{C \times \frac{H}{2} \times \frac{W}{2}}$ that encodes the events in input using $C = 64$ filters.

4.2.2 Transformer Encoder

In order to create a sequence of tokens for the transformer encoder, as shown in Figure 4.3, the feature maps F extracted by the convolutional encoder are divided into patches of equal size $P \times P$, which was set experimentally to 32×32, obtaining a total number of patches $N_p = N_y \times N_x = \frac{H/2}{P} \times \frac{W/2}{P}$ for each channel. Afterwards, those patches are unrolled over the filters dimension and flattened obtaining a tensor $F_{tokens} \in \mathbb{R}^{CN_p \times P^2}$. As described in the original Transformer paper [48], sine

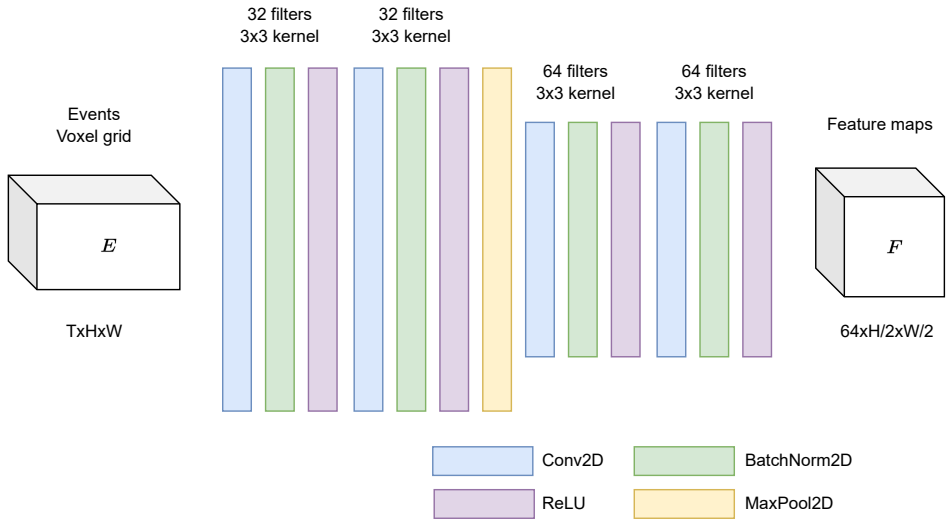

Figure 4.2. Architecture of the convolutional encoder.

and cosine positional encoding vectors are then added to F_{tokens} in order to inject positional information, and the sequence is self-attended by the Transformer Encoder, producing as output an equally-shaped tensor $\hat{F}_{tokens}$ which is now enriched with global context embeddings. Note that due to the choice of fixed sized patches, half resolution of the input sensor size must be divisible by 32, but it is possible to pad the input tensor as it will be done in Section 5.3.1.

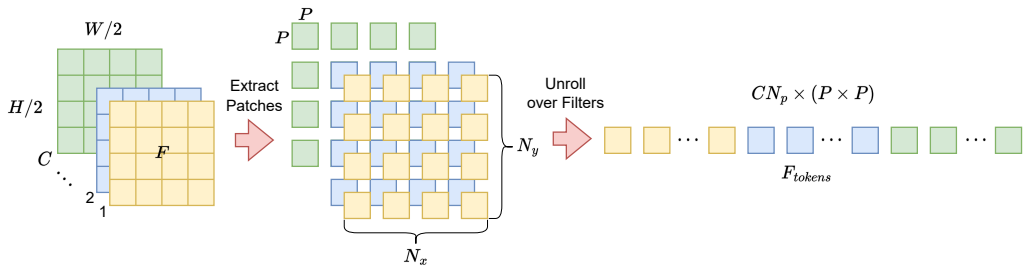

Figure 4.3. Procedure to turn feature maps into a sequence of tokens for the Transformer encoder.

4.2.3 Convolutional Decoder

The final module of the architecture is a convolutional decoder that takes as input the enriched feature maps and decodes them into an RGB image $\hat{I} \in [0,1]^{3\times H\times W}$. In order to do that it is necessary to transform $\hat{F}_{tokens}$ into a tensor resembling the feature maps, which can be done by recomposing the patches following the previous procedure backwards. This means that the shape of $\hat{F}_{tokens} \in \mathbb{R}^{CN_p\times P^2}$ is first expanded into $C \times (N_y \times N_x) \times (P \times P)$, then rearranging the dimensions is transformed into $C \times (N_y \times P) \times (N_x \times P)$, and finally by reassembling the patches is obtained $Z \in \mathbb{R}^{C\times H/2\times W/2}$ that can be processed by the decoder.

The architecture of the module, shown in Figure 4.4, is specular to the one of the encoder, employing transposed convolutional layers instead of regular ones. In order to produce valid RGB images as output, there is a final convolutional layer with three filters to create RGB channels, and the sigmoid activation function, to produce values in the range $[0, 1]$.

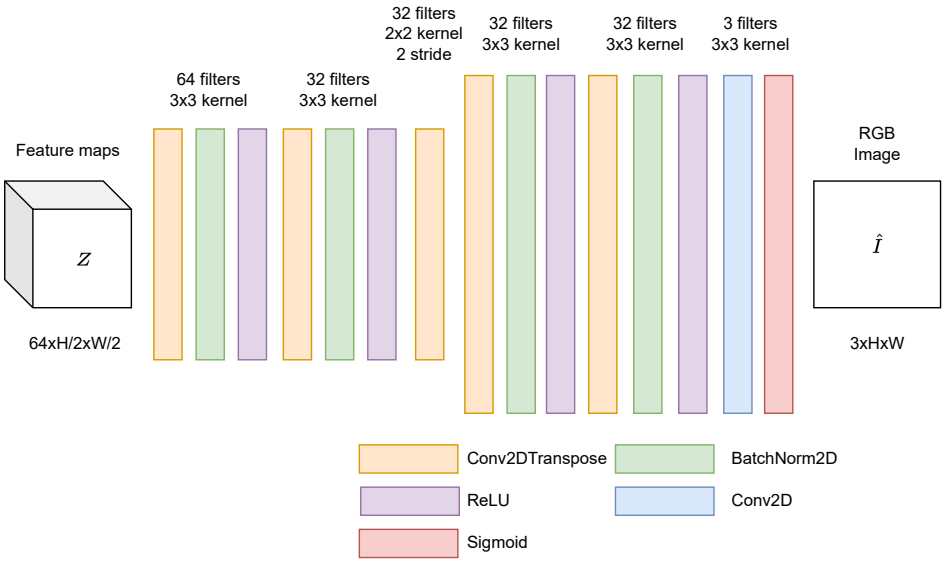

Figure 4.4. Architecture of the convolutional decoder.

4.3 Training Loss

In order to train the model, a weighted sum of two loss terms has been used. The first one, called *image loss* L_i, is the term that indicates the model to reconstruct images as similar as possible to the ground truth and consists in the Mean Squared Error (MSE), computed between the ground truth image I and the one produced by the model $\hat{I}$:

$$L_i(I, \hat{I}) = \frac{1}{3 \times H \times W} \sum_{i=1}^{3} \sum_{j=1}^{H} \sum_{k=1}^{W} (I[i,j,k] - \hat{I}[i,j,k])^2 \tag{4.3}$$

The second one instead, is called *similarity loss* L_s and introduces robustness in the model by forcing the transformer to output features similar to the one produced by the CNN. The term consists in the MSE between the local features F and the ones obtained after the transformer and reshaping Z:

$$L_s(F, Z) = \frac{1}{C \times H/2 \times W/2} \sum_{i=1}^{C} \sum_{j=1}^{H/2} \sum_{k=1}^{W/2} (F[i,j,k] - Z[i,j,k])^2 \tag{4.4}$$

The weighted sum formula is therefore:

$$Loss = w_1 L_i(I, \hat{I}) + w_2 L_s(F, Z) \tag{4.5}$$

Chapter 5

Evaluation and Experiments

In this chapter, implementation details of the model will be discussed in Section 5.1, while the methodology to create the synthetic dataset will be presented in Section 5.2. Instead, in Section 5.3 are reported the results obtained with the proposed model. Finally in Section 5.4 is reported an experimental study on two different different kind of architectures which have been tested during the thesis.

5.1 Implementation

The model and all the experiments have been implemented on the PyTorch framework [31] and using the PyTorch Lightning library [12]. In order to track the different experiments, the TensorBoard toolkit [8] has been employed, allowing to compare visually metrics acquired from different runs and also enabling to inspect image reconstruction progress over the training epochs.

Training has been performed locally on a NVIDIA GTX 1060 3GB and remotely on the Google Colab platform which freely offers NVIDIA K80s, P100s, P4s, T4s according to their usage policies.

The final model uses $w_1 = 1$ and $w_2 = 0.01$ as weights for the loss function and has been trained with the Adam optimizer for 1000 epochs, using $1e-4$ as learning rate and a batch size equal to 64. Finally, the Transformer encoder is composed of a single layer, having 4 heads and 2048 as feed forward embedding size since increasing those numbers would lead to overfitting the training data due to the relatively small size of the dataset and the large amount of parameters which is about 8.5 million.

5.2 ESIM and Synthetic Dataset

Currently the only dataset available that offers color events and the corresponding ground truth images is the CED dataset [43]. However the images provided are not exempt from regular cameras defects like motion blur and low dynamic range, therefore it's not advisable to use them as ground truth, since the model would suffer from the same limitations when reconstructing images from events. This is why many of the works in the state of art, including this one, use a synthetic dataset generated from an event cameras simulator.

In the particular case of this work, the ESIM simulator [35] was used, which adopts an adaptive sampling scheme in order to generate events following their asynchronous nature. With respect to previous simulators, ESIM doesn't use a fixed-sampling scheme which means that it can reliably simulate event data even when the brightness signal changes more rapidly than the rendering framerate. This has been achieved through a tight implementation between the rendering engine and the event simulator that can adaptively query frames based on the dynamics of the visual signal. The simulator also implements a noise model for event cameras that can vary the contrast threshold using a normal distribution, enabling also to set different limits for the positive and negative contrast thresholds.

Moreover, ESIM offers multiple rendering engines that allow to produce events from different media sources. To generate the synthetic dataset used in this thesis, the PlanarRenderer engine has been used, which projects a 2D image onto a plane in a 3D space and renders a virtual camera. In order to record an event stream, the camera is moved along a random trajectory, simulating events generated by the brightness changes caused by the texture.

As a source for the images, the DIV2K dataset has been selected, which is typically used for Super-Resolution and image restoration tasks, providing 1000 diverse images at 2K resolution, much higher than typical image datasets. The choice is motivated by the fact that the virtual camera shows only a portion of the images in order to be able to move around, so bigger images are needed in order to obtain high quality ground truths.

The simulator has been set up with the standard configuration by changing the parameters reported in Table 5.1, generating for each image in the dataset a sequence of events and about 5 ground truth images.

Parameter Name	**Value**
random_seed	random for each sequence
camera_trajectory_length	0.2s
camera_trajectory_sampling_frequency	10Hz
publisher_frame_rate	30 fps

Table 5.1. ESIM parameters changed from standard configuration.

The simulated sequences are stored in the rosbag format which resulted in slow reading speeds using the rosbag Python library, so the dataset has been preprocessed by producing event grids with $T = 10$, which are stored on the disk to reduce the training time. Each training pair is composed by a timestamped image and an event grid, generated by considering all the events between the previous image timestamp and the new one, so in this case N_e is variable. Moreover the events sequences and the corresponding images have been cropped to 128×128 resolution by taking the top left square in order to make training faster. Finally the dataset has been split in 800 sequences used for training, and 200 for test. Some visual examples of the content of the dataset can be seen in Figure 5.1.

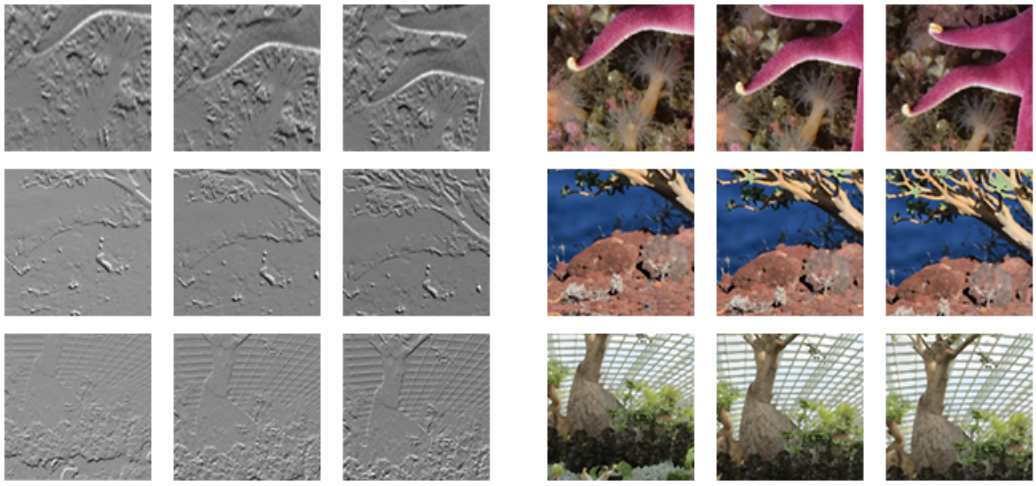

Figure 5.1. Some examples of sequences generated in the synthetic dataset. On the left some events are shown in the form of slices of an event voxel grid, while on the right there are the corresponding ground truth images.

5.3 Results

In order to evaluate numerically the results of the proposed method, three different metrics has been used to compare the ground truth image I with the one predicted by the model $\hat{I}$:

- **MSE**: measures the Mean Squared Error between the pixel values of the two images;

$$MSE(I, \hat{I}) = \frac{1}{3 \times H \times W} \sum_{i=1}^{3} \sum_{j=1}^{H} \sum_{k=1}^{W} (I[i,j,k] - \hat{I}[i,j,k])^2 \tag{5.1}$$

- **SSIM**: is the Structure Similarity Index Measure and is used to evaluate the perceived quality of digital pictures by leveraging local spatial inter-dependencies, analyzing the structure of an image. In order to do so, the index is computed on 11×11 windows according to the following formula:

$$\frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.2}$$

  In this case, x is a window of the target image I, while y is a window of the predicted image $\hat{I}$; μ_x, μ_y are the mean of the pixels contained in the respective windows, σ_x, σ_y are their variance and σ_{xy} is the covariance of x and y; $c1$ and $c2$ are two variables used to stabilize the division and for images in the range [0, 1], they are set respectively to 0.01^2 and 0.03^2;

- **LPIPS**: stands for Learned Perceptual Image Patch Similarity and has been introduced in [57]. It is the metric that resembles most the human perception of similarities between images and is measured by computing the MSE between

the features extracted from both images by using a VGG-19 network pre-trained on the ImageNet dataset.

The quantitive results are summarized in Table 5.2 while qualitative results are shown in Figure 5.2.

As can be seen, the model is able to reconstruct the majority of the edges with good fidelity, hence the 60% SSIM, while colors tend to not match perfectly the ground truth image, respecting however the brightness changes. Moreover, the model is also capable of allucinating some parts of the images where there aren't too many events available, although details are not very defined.

MSE	SSIM	LPIPS
0.0416	0.6012	0.4745

Table 5.2. Quantitative results of the model on the synthetic test dataset.

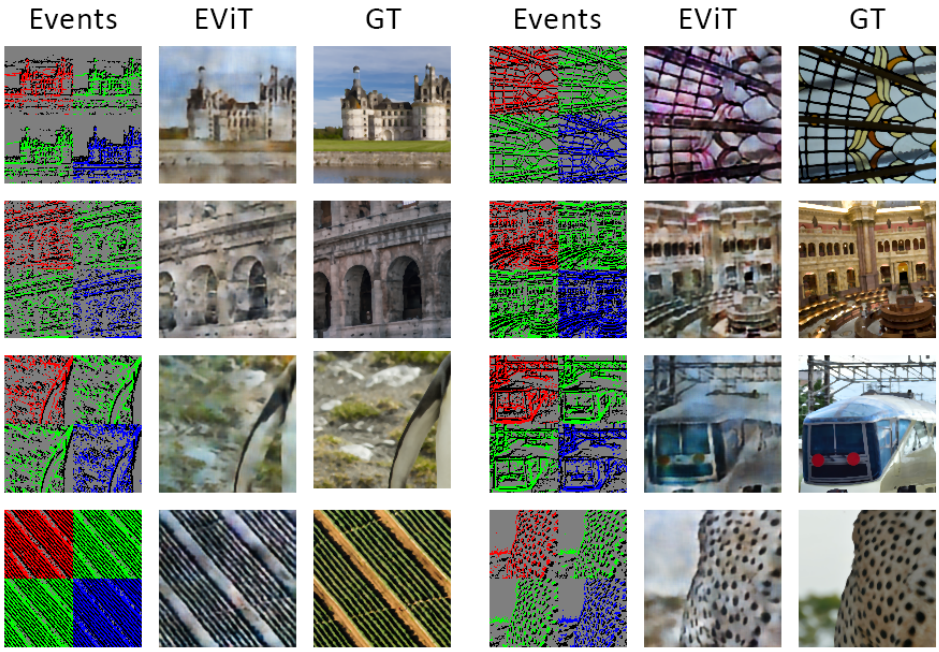

Figure 5.2. Qualitative results obtained from the model on the synthetic test set. The *Events* column represents the middle temporal bin of the input voxel grid, splitting the RGGB channels and representing with colors positive polarity while with black negative polarity. In the *EViT* column are reported the images produced by the model, while under the *GT* column are depicted the ground truth images used as reference.

5.3.1 State-of-the-art comparison

The results presented until now are computed on a reserved part of the synthetic dataset generated, but it is also possible to analyze how well the model generalizes on real event data. In order to do so, the model is evaluated on the CED Dataset, which is the first and currently the only dataset that contains footage with both color events and the corresponding images. The dataset contains 50 minutes of footage coming from different scenes which are shot both in indoor and outdoor environments in order to highlight event cameras strengths, i.e. high dynamic range and resilience to motion blur. The sequences are captured using the Color-DAVIS346 event camera which mounts a 8×6mm CMOS chip that produces color events and standard frames at 346×260 resolution, which however suffer from the previously mentioned defects since they are captured by normal APS sensors. In order to provide color information, the chip is overlayed with an RGGB Bayer pattern which makes some pixels more sensitive to certain wavelengths, and to obtain color images, a demosaicing algorithm is used on the raw measurements. The dataset provides the data in the form of binary rosbag files that contain events synchronized with raw and color images with microsecond timestamp precision.

In order to run the model proposed on this dataset, some additional adaptation is required in the architecture, since it needs to extract 32×32 patches from the feature maps which have half resolution. Therefore, the voxel grids in input are padded with zeros in order to form a 3D volume with the smallest shape multiple of $2 * 32$, that in this case is $10 \times 320 \times 384$. The output produced by the model has the same resolution and consequentially it is cropped to the original resolution of 346×260.

As a comparison with the state of the art, the E2VID model [37] has been chosen since it's currently one of the best models in the task of color image reconstruction and conveniently its code is available on GitHub along with pretrained weights. The results, shown in Figure 5.3, have been obtained by using the default parameters of the provided command line tool by specifying the option to enable the reconstruction of colored events. In their default configuration, N_e is set to $0.35 * H * W$, so the same number of events has been used, producing voxel grid using 31.486 events.

From the images, it can be seen that the model proposed is again able to identify and reconstruct edges but it has problems to reconstruct colors. In particular, in the last row is reported an interesting scenario that shows how the two models react when no events are available for parts of the image, since in this case the camera is fixed and there are only people dancing. By looking at the results, it seems that the model proposed can't handle very well the parts in which events are too sparse, probably due to the fact that the sequences contained in the synthetic dataset on which it has been trained are much denser of events. In fact, it is possible to inspect the average sparseness of the event grids produced from both datasets, by computing the ratio of voxels containing zeros over the total number of voxels. The results are that event grids generated from the CED Dataset have an average sparseness of 91%, in contrast to the 64% obtained from those produced from the synthetic dataset.

Figure 5.3. Visual comparison on the CED Dataset between the model proposed and E2VID [37].

5.4 Different model architectures

The architecture presented in Section 4.2 was not the only one tested during the thesis, but it is the one that achieved the best results. In the next subsections, two different approaches to image reconstruction will be presented: a greyscale model and a teacher-student model. Finally, the respective results will be compared with the EViT model previously described.

5.4.1 Greyscale model

Since the model doesn't manage to reconstruct well colors but is capable of reconstructing their brightness, inspired by some works like [37, 51, 30], an attempt was made to split the color events into three grey scale channels, reconstructing separately three grey scale images and then fusing them in order to get an RGB image as output.

In particular, the architecture has been kept unchanged apart from the last convolutional layer of the convolutional decoder which was set to 1 output filter in order to obtain single channel grey scale images.

A new version of the synthetic dataset has been generated using the ESIM simulator, this time emitting grey scale events, and the images have been converted into greyscale values as specified by the BT.601 standard, i.e. by computing the dot product between the images and the vector $[0.2989, 0.5870, 0.1140]$. After that, the model has been trained following the same procedure described before using the greyscale dataset.

At inference time, in order to output RGB images, color events are taken as input and the produced voxel grids are downsampled to half resolution by splitting them according to the Bayer RGGB filter which is put in front of the simulated camera. In particular, from a color event voxel grid E, the single channel voxel grids $R, G, B \in \mathbb{R}^{H/2 \times W/2}$ are obtained in the following way:

$$\begin{aligned} \mathrm{R}[x, y, t] =& E[2x, 2y, t] \\ \mathrm{G}[x, y, t] =& \frac{E[2x+1, 2y, t] + E[2x, 2y+1, t]}{2} \\ \mathrm{B}[x, y, t] =& E[2x+1, 2y+1, t] \end{aligned} \tag{5.3}$$

The three voxel grids are then given as input to the model which will produce as output three grey scale images $\hat{R}, \hat{G}, \hat{B} \in \mathbb{R}^{H/2 \times W/2}$. Those images are finally upscaled to the original $H \times W$ resolution using bilinear interpolation, and they are stacked in order to obtain a final image $\hat{I} \in \mathbb{R}^{3 \times H \times W}$.

5.4.2 Teacher-Student model

Another approach that was tested is the teacher-student technique, in which a *teacher* model helps another model, called *student*, to learn a complex task. In the case of the image reconstruction task, CNNs are still the neural network architecture that showed the best results, so trying to distill some knowledge from CNNs to a Transformer architecture was part of an experimental study.

Therefore, a teacher architecture based on a Convolutional Auto-Encoder was created, composed of an encoder, which extracts the most significant features that can represent an image of the dataset (*code*), and a decoder, which reconstructs the original image starting from the features. In the particular architecture used, shown in Figure 5.4, the code is a tensor $z \in \mathbb{R}^{128 \times 8 \times 8}$.

The student architecture instead is based on Transformers and is shown in Figure 5.5. It is again inspired by the ViT [9] architecture but this time it operates directly on the voxel grids. In fact, the first part of the model extracts 16×16 patches from the voxel grids, obtaining a total number of 8×8 patches, and linearly projects them with the temporal bins into a 128 dimensional space ($f : \mathbb{R}^{TP^2} \to \mathbb{R}^{128}$). The obtained tokens are then self-attended by a transformer encoder which learns the needed features to encode the voxel grids. The resulting features can be then decoded using the pretrained decoder of the convolutional teacher in order to obtain RGB images as output.

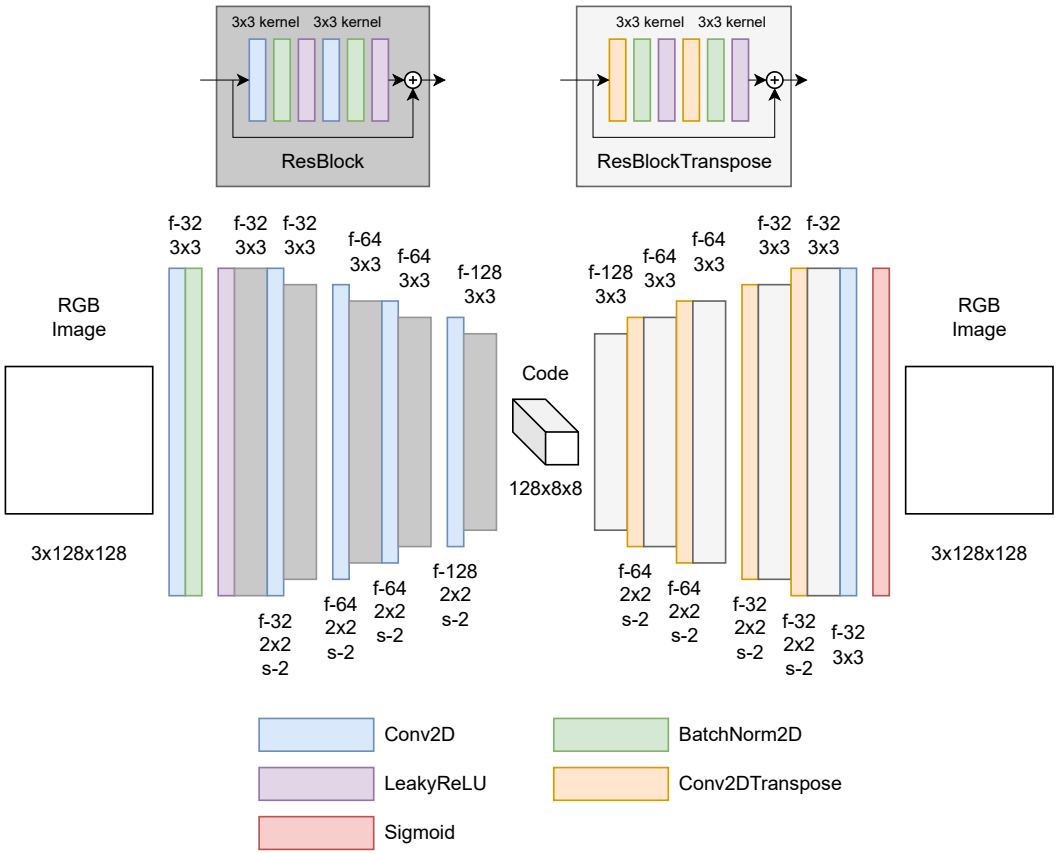

Figure 5.4. Architecture of the Teacher model which is a Convolutional Auto-Encoder. On convolutional layers, f indicates the number of filters and s the stride.

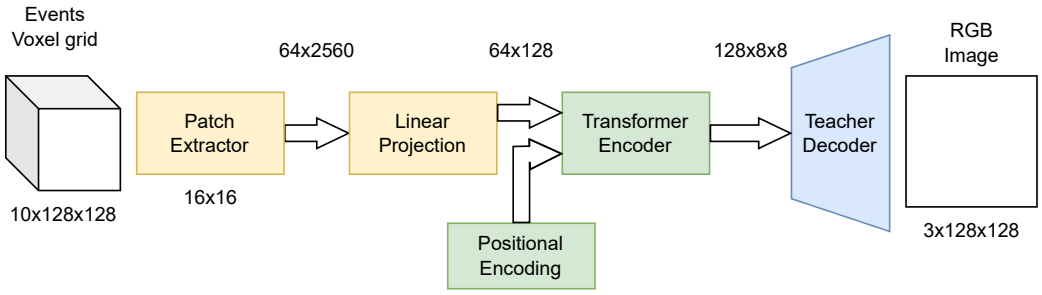

Figure 5.5. Architecture of the Student model based on Transformers.

5.4.3 Comparison

The results of the architectures discussed are compared with the best model in Figure 5.6 and even if they do not improve the performances, it is interesting to observe them.

For what concerns the results of the student, it can be seen that there are major inconsistencies in the smoothness of the images, since the convolutional decoder of the teacher is quite sensible to small changes in the 8×8 feature maps, causing abrupt discontinuities from one patch to the other.

Instead the black and white model, while not suffering from visible patches inconsistencies, still cannot obtain results with good color accuracy, showing colored halos all over the images.

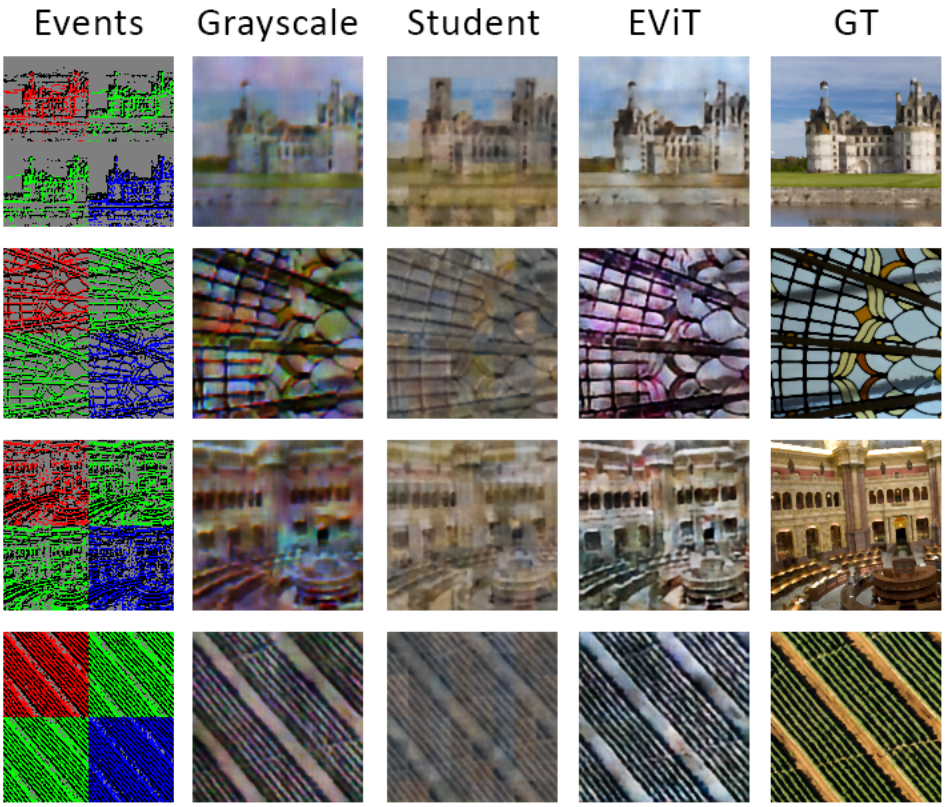

Figure 5.6. Visual comparison of the results of the Grayscale and Teacher-Student architectures with the EViT.

Chapter 6

Conclusions

In this final chapter some conclusions are drawn in Section 6.1 and some possible future works are proposed in Section 6.2.

6.1 Final conclusions

In this thesis it has been proposed EViT, the first deep learning model that performs the image reconstruction task starting from event cameras working directly with color events. Since event cameras introduce a new paradigm to capture the environment without motion blur and with high dynamic range, it is useful to transform the produced events into the more familiar image data that can be further processed using any out-of-the-box approach to perform Computer Vision downstream tasks. The architecture proposed is based on CNNs to extract local features from events turned into a voxel grid, and the Transformer architecture that correlates them capturing global correlations. In particular, a CNN encoder extracts local features from the events which are divided into patches to create a sequence for the Transformer. The sequence is then self-attended to output global features which are finally used by a CNN decoder to generate RGB images. In order to train the model, a synthetic dataset has been generated using the ESIM simulator and DIV2K dataset images, which are projected into a 3D space and a virtual moving camera generates events framing the images. The model has then been tested on a reserved part of the synthetic dataset showing good results, especially on reconstructing the edges. As a study, the generalization ability of the model has also been tested on a real dataset comparing the results with a state-of-the-art architecture. Two other experiments have additionally been performed in order to try to improve the results, by using a teacher-student approach and by reconstructing one grayscale image for each RGB channels to finally fuse them.

6.2 Future works

Some possible future works could improve the generalization capability of the model for real data since it has some problems as shown by the results on the CED dataset. For example, it would be possible to perform data augmentation on the training set,

by adding some noise to the incoming events or by generating synthetic sequences setting different combinations of contrast thresholds.

Moreover, the actual architecture reconstructs a single image, so in the future it would be possible to output a sequence of coherent frames in order to form a video that can be processed or visualized in real time. This could be done by training the model with an additional temporal loss which is also used in some other works in the state of art, making consecutive frames more similar among each other.

Bibliography

[1] Aelmore, S., Ordonez, R. C., Parameswaran, S., and Mauger, J. Real-time event-based tracking and detection for maritime environments (2022). Available from: https://arxiv.org/abs/2202.04231, doi:10.48550/ARXIV.2202.04231.

[2] Agustsson, E. and Timofte, R. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2017).

[3] Bardow, P., Davison, A. J., and Leutenegger, S. Simultaneous optical flow and intensity estimation from an event camera. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 884–892 (2016). doi:10.1109/CVPR.2016.102.

[4] Brandli, C., Berner, R., Yang, M., Liu, S.-C., and Delbruck, T. A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, **49** (2014), 2333. doi:10.1109/JSSC.2014.2342715.

[5] Chamorro, S., Collier, J., and Grondin, F. Neural network based lidar gesture recognition for realtime robot teleoperation. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE (2021). Available from: https://doi.org/10.1109%2Fssrr53300.2021.9597855, doi:10.1109/ssrr53300.2021.9597855.

[6] Chen, G., Xu, Z., Li, Z., Tang, H., Qu, S., Ren, K., and Knoll, A. A novel illumination-robust hand gesture recognition system with event-based neuromorphic vision sensor. *IEEE Transactions on Automation Science and Engineering*, **18** (2021), 508.

[7] Cook, M., Gugelmann, L., Jug, F., Krautz, C., and Steger, A. Interacting maps for fast visual interpretation. In *The 2011 International Joint Conference on Neural Networks*, pp. 770–776 (2011). doi:10.1109/IJCNN.2011.6033299.

[8] Developers, T. Tensorflow (2022). Available from: https://doi.org/10.5281/zenodo.6574269, doi:10.5281/zenodo.6574269.

[9] Dosovitskiy, A., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, **abs/2010.11929** (2020). Available from: https://arxiv.org/abs/2010.11929, arXiv:2010.11929.

[10] Duwek, H. C., Shalumov, A., and Tsur, E. E. Image reconstruction from neuromorphic event cameras using laplacian-prediction and poisson integration with spiking and artificial neural networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1333–1341 (2021). doi:10.1109/CVPRW53098.2021.00147.

[11] Esser, S. K., et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, **113** (2016), 11441. Available from: https://www.pnas.org/doi/abs/10.1073/pnas.1604850113, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1604850113, doi:10.1073/pnas.1604850113.

[12] Falcon, W., et al. Pytorchlightning/pytorch-lightning: 0.7.6 release (2020). Available from: https://doi.org/10.5281/zenodo.3828935, doi:10.5281/zenodo.3828935.

[13] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36** (2004), 193.

[14] Gallego, G., Rebecq, H., and Scaramuzza, D. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3867–3876 (2018). doi:10.1109/CVPR.2018.00407.

[15] Ghosh, R., Mishra, A., Orchard, G., and Thakor, N. V. Real-time object recognition and orientation estimation using an event-based camera and cnn. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pp. 544–547 (2014). doi:10.1109/BioCAS.2014.6981783.

[16] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Commun. ACM*, **63** (2020), 139–144. Available from: https://doi.org/10.1145/3422622, doi:10.1145/3422622.

[17] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, **abs/1512.03385** (2015). Available from: http://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[18] I., S. M. M., Choi, J., and Yoon, K. Learning to super resolve intensity images from events. *CoRR*, **abs/1912.01196** (2019). Available from: http://arxiv.org/abs/1912.01196, arXiv:1912.01196.

[19] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, **abs/1502.03167** (2015). Available from: http://arxiv.org/abs/1502.03167, arXiv:1502.03167.

[20] Islam, A., Asikuzzaman, M., Khyam, M. O., Noor-A-Rahim, M., and Pickering, M. R. Stereo vision-based 3d positioning and tracking. *IEEE Access*, **8** (2020), 138771. doi:10.1109/ACCESS.2020.3011360.

[21] Jiao, J., Huang, H., Li, L., He, Z., Zhu, Y., and Liu, M. Comparing representations in tracking for event camera-based slam (2021). Available from: https://arxiv.org/abs/2104.09887, doi:10.48550/ARXIV.2104.09887.

[22] Kolb, A., Barth, E., Koch, R., and Larsen, R. Time-of-flight sensors in computer graphics. *Proc. Eurographics (State Art Re.)*, **2009** (2008).

[23] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, p. 1097–1105. Curran Associates Inc., Red Hook, NY, USA (2012).

[24] Kueng, B., Mueggler, E., Gallego, G., and Scaramuzza, D. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 16–23 (2016). doi:10.1109/IROS.2016.7758089.

[25] Lichtsteiner, P., Posch, C., and Delbruck, T. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, **43** (2008), 566. doi:10.1109/JSSC.2007.914337.

[26] Liu, M. and Delbruck, T. Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. In *British Machine Vision Conference (BMVC) 2018*. BMVC, Newcastle upon Tyne, UK (2018). Available from: https://doi.org/10.5167/uzh-168589.

[27] Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, **10** (1997), 1659. Available from: https://www.sciencedirect.com/science/article/pii/S0893608097000117, doi:https://doi.org/10.1016/S0893-6080(97)00011-7.

[28] McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, **5** (1943), 115. Available from: https://doi.org/10.1007/bf02478259, doi:10.1007/bf02478259.

[29] Mondal, A., R, S., Giraldo, J. H., Bouwmans, T., and Chowdhury, A. S. Moving object detection for event-based vision using graph spectral clustering. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. IEEE (2021). Available from: https://doi.org/10.1109%2Ficcvw54120.2021.00103, doi:10.1109/iccvw54120.2021.00103.

[30] Mostafavi, M., Nam, Y., Choi, J., and Yoon, K.-J. E2sri: Learning to super-resolve intensity images from events. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44** (2022), 6890. doi:10.1109/TPAMI.2021.3096985.

[31] Paszke, A., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates,

Inc. (2019). Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[32] Pini, S., Borghi, G., Vezzani, R., and Cucchiara, R. Learn to see by events: RGB frame synthesis from event cameras. *CoRR*, **abs/1812.02041** (2018). Available from: http://arxiv.org/abs/1812.02041, arXiv:1812.02041.

[33] Pini, S., Borghi, G., Vezzani, R., and Cucchiara, R. Video synthesis from intensity and event frames. In *Image Analysis and Processing – ICIAP 2019: 20th International Conference, Trento, Italy, September 9–13, 2019, Proceedings, Part I*, p. 313–323. Springer-Verlag, Berlin, Heidelberg (2019). ISBN 978-3-030-30641-0. Available from: https://doi.org/10.1007/978-3-030-30642-7_28, doi:10.1007/978-3-030-30642-7_28.

[34] Posch, C., Matolin, D., and Wohlgenannt, R. A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE Journal of Solid-State Circuits*, **46** (2011), 259. doi:10.1109/JSSC.2010.2085952.

[35] Rebecq, H., Gehrig, D., and Scaramuzza, D. Esim: an open event camera simulator. In *Proceedings of The 2nd Conference on Robot Learning* (edited by A. Billard, A. Dragan, J. Peters, and J. Morimoto), vol. 87 of *Proceedings of Machine Learning Research*, pp. 969–982. PMLR (2018). Available from: https://proceedings.mlr.press/v87/rebecq18a.html.

[36] Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. Events-to-video: Bringing modern computer vision to event cameras. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3852–3861 (2019). doi:10.1109/CVPR.2019.00398.

[37] Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. High speed and high dynamic range video with an event camera. *CoRR*, **abs/1906.07165** (2019). Available from: http://arxiv.org/abs/1906.07165, arXiv:1906.07165.

[38] Reinbacher, C., Graber, G., and Pock, T. Real-time intensity-image reconstruction for event cameras using manifold regularisation. *CoRR*, **abs/1607.06283** (2016). Available from: http://arxiv.org/abs/1607.06283, arXiv:1607.06283.

[39] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, **abs/1505.04597** (2015). Available from: http://arxiv.org/abs/1505.04597, arXiv:1505.04597.

[40] Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, **11** (2017). Available from: https://www.frontiersin.org/articles/10.3389/fnins.2017.00682, doi:10.3389/fnins.2017.00682.

[41] Scheerlinck, C., Barnes, N., and Mahony, R. E. Continuous-time intensity estimation using event cameras. *CoRR*, **abs/1811.00386** (2018). Available from: http://arxiv.org/abs/1811.00386, arXiv:1811.00386.

[42] Scheerlinck, C., Rebecq, H., Gehrig, D., Barnes, N., Mahony, R. E., and Scaramuzza, D. Fast image reconstruction with an event camera. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 156–163 (2020). doi:10.1109/WACV45572.2020.9093366.

[43] Scheerlinck, C., Rebecq, H., Stoffregen, T., Barnes, N., Mahony, R. E., and Scaramuzza, D. CED: color event camera dataset. *CoRR*, **abs/1904.10772** (2019). Available from: http://arxiv.org/abs/1904.10772, arXiv:1904.10772.

[44] Serrano-Gotarredona, R., et al. Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing– learning–actuating system for high-speed visual object recognition and tracking. *IEEE Transactions on Neural Networks*, **20** (2009), 1417. doi:10.1109/TNN.2009.2023653.

[45] Shaikh, M. B. and Chai, D. Rgb-d data-based action recognition: A review. *Sensors*, **21** (2021). Available from: https://www.mdpi.com/1424-8220/21/12/4246, doi:10.3390/s21124246.

[46] She, C. and Qing, L. Spikeformer: Image reconstruction from the sequence of spike camera based on transformer. In *2022 the 5th International Conference on Image and Graphics Processing (ICIGP)*, ICIGP 2022, p. 72–78. Association for Computing Machinery, New York, NY, USA (2022). ISBN 9781450395465. Available from: https://doi.org/10.1145/3512388.3512399, doi:10.1145/3512388.3512399.

[47] Shi, X., Chen, Z., Wang, H., Yeung, D., Wong, W., and Woo, W. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, **abs/1506.04214** (2015). Available from: http://arxiv.org/abs/1506.04214, arXiv:1506.04214.

[48] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, **abs/1706.03762** (2017). Available from: http://arxiv.org/abs/1706.03762, arXiv:1706.03762.

[49] Wang, L., Kim, T., and Yoon, K. Eventsr: From asynchronous events to image reconstruction, restoration, and super-resolution via end-to-end adversarial learning. *CoRR*, **abs/2003.07640** (2020). Available from: https://arxiv.org/abs/2003.07640, arXiv:2003.07640.

[50] Wang, L., Kim, T.-K., and Yoon, K.-J. Joint framework for single image reconstruction and super-resolution with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44** (2022), 7657. doi:10.1109/TPAMI.2021.3113352.

[51] Wang, L., Kim, T.-K., and Yoon, K.-J. Joint framework for single image reconstruction and super-resolution with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44** (2022), 7657. doi:10.1109/TPAMI.2021.3113352.

[52] Wang, Q., Zhang, Y., Yuan, J., and Lu, Y. Space-time event clouds for gesture recognition: From rgb cameras to event cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1826–1835. IEEE (2019).

[53] Wang, Z., Ng, Y., Scheerlinck, C., and Mahony, R. E. An asynchronous kalman filter for hybrid event cameras. *CoRR*, **abs/2012.05590** (2020). Available from: https://arxiv.org/abs/2012.05590, arXiv:2012.05590.

[54] Weikersdorfer, D., Adrian, D. B., Cremers, D., and Conradt, J. Event-based 3d slam with a depth-augmented dynamic vision sensor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 359–364 (2014). doi:10.1109/ICRA.2014.6906882.

[55] Weng, W., Zhang, Y., and Xiong, Z. Event-based video reconstruction using transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2543–2552 (2021). doi:10.1109/ICCV48922.2021.00256.

[56] Zhang, C., Zhang, N., Ma, Z., Wang, L., Qin, Y., Jia, J., and Zang, K. A 240 × 160 3d-stacked spad dtof image sensor with rolling shutter and in-pixel histogram for mobile devices. *IEEE Open Journal of the Solid-State Circuits Society*, **2** (2022), 3. doi:10.1109/OJSSCS.2021.3118332.

[57] Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, **abs/1801.03924** (2018). Available from: http://arxiv.org/abs/1801.03924, arXiv:1801.03924.

[58] Zhu, A., Yuan, L., Chaney, K., and Daniilidis, K. EV-FlowNet: Self-supervised optical flow estimation for event-based cameras. In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation (2018). Available from: https://doi.org/10.15607%2Frss.2018.xiv.062, doi:10.15607/rss.2018.xiv.062.

[59] Zhu, L., Wang, X., Chang, Y., Li, J., Huang, T., and Tian, Y. Event-based video reconstruction via potential-assisted spiking neural network. *CoRR*, **abs/2201.10943** (2022). Available from: https://arxiv.org/abs/2201.10943, arXiv:2201.10943.