Fill In The Gaps (FIG): Object Localization and Tracking for Event Based Cameras

Richard Chen [laurelin@seas], Wesley Penn [wespenn@seas], Bhaskar Sen [bsen26@seas], Suryansh Loya [suryansh@wharton]

***Abstract*—Object localization and tracking using event-based cameras is a promising approach for real-time object detection in autonomous systems. In this paper, we present a framework for converting sequential event streams from event-based cameras into grid-based representations using bucketing and propose a network architecture specifically designed to capture the temporal features present in event-based camera data. We show that even though object detection is slightly worse than conventional frame cameras, it can be very useful for tracking objects between frames and can be applied to low latency situations such as autonomous cars or low lighting conditions.**

I. Introduction

We aim to perform object localization and tracking using Event Based Cameras in order to fill in the information gaps between traditional RGB camera frames. Event Based Cameras (also called Event Cameras, Spiking Cameras) are a biologically inspired camera architecture. [1] Instead of capturing images at a fixed rate, they asynchronously measure per-pixel brightness changes. The advantages of these cameras include high dynamic range, low latency, high temporal resolution, and performance in adverse lighting conditions. These strengths of Event Based Cameras cover traditional weaknesses of Frame Camera, which are normal video cameras that capture data at a fixed rate, taking in the whole scene as an image.

Object Localization detects the main object in an image, while Object Tracking finds the location of the object through time[2]. These tasks can be useful for situations where rapid object detection is necessary, such as autonomous vehicles, which need real-time information about the environment very low latency. By combining bounding box detection with event-based cameras, it is possible to quickly and accurately identify and track objects in the environment, which can be used to help the autonomous vehicle make decisions about how to navigate and avoid collisions.

While Event Based Cameras are currently a novel technology, their strengths give them a variety of potential applications. Because of this, we investigate the performance of Event Based Cameras in traditional computer vision tasks.

A. Contributions

In this work, we

- Introduce a framework for converting sequential event streams into grid-based representations using bucketing, to allow for the efficient processing
- Propose a network architecture that is specifically designed to capture the temporal features present in event-based camera data
- Mention the issues related to object detection using event-based camera data, including the difficulties posed by static objects and the sparse nature of the data

II. Background

In this case, we can reduce the tracking problem down into a series of localization problems.

Unlike traditional video frames, the data from the Event Based Camera is an asynchronous stream of events. Each event consists of $[t, x, y, \Delta]$ with Δ representing the sign of the polarity (change in brightness).

Two common approaches to working with this type of data involve either treating as video frames or just as-is. The first approach groups the data into temporal buckets to simulate camera frames. The second approach deals with the event stream directly. The former lends itself better to classical computer vision techniques, since the event data is transformed to act like video frames. However, this comes with some of the same drawbacks of frame cameras. The other approach works with the event stream directly. While this can take full advantage of all the benefits of event cameras, asynchronous data streams are different from synchronous video frames. Due to this, not all techniques have a direct analogue.

To detect and track objects using event camera data, a neural network can be developed to identify patterns of events that correspond to the presence of an object. These patterns could include a cluster of events at a specific location, or a series of events that follow a particular trajectory. Once an object has been detected, a bounding box can be placed around it to denote its location and size. Bounding boxes are commonly used in object detection tasks to enclose objects of interest and can be useful for further analysis or manipulation.

III. Related Work

A. Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a type of neural network that are particularly well-suited for image and video recognition tasks. CNNs are composed of multiple layers of convolutional filters, which are used to extract features from the input data. These features are then processed by additional layers of the network, which can be used to classify the input data or perform other tasks.

Since their introduction in the late 1980s, CNNs have been applied to a wide range of tasks, including object recognition, image classification, and face detection. One of the key advantages of CNNs is their ability to learn features directly from the input data, rather than requiring hand-crafted features. This has made CNNs a popular choice for tasks where hand-crafted features may be difficult to design, such as in image and video recognition tasks.

There have been a number of efforts to improve the performance and efficiency of CNNs. For example, research has focused on developing techniques for training deeper networks, as well as on developing architectures that can achieve better performance with fewer parameters. Additionally, there has been a significant amount of work on using CNNs in combination with other types of neural networks, such as recurrent neural networks, to improve performance on a wide range of tasks.

In recent years, CNNs have become the dominant approach for tasks such as image classification and object recognition, and they have achieved state-of-the-art performance on a number of benchmarks. In our task for event based camera data, we use CNNs powerful architecture to extract a rich set of features from the event based camera data.

B. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

Recurrent neural networks (RNNs) are a type of neural network that are particularly well-suited for processing sequential data, such as time series or natural language. RNNs have the ability to ”remember” past input and use this information to inform the processing of future input, making them a powerful tool for tasks such as language translation and speech recognition.

One type of RNN that has gained widespread attention is the long short-term memory (LSTM) network. LSTMs were introduced by Hochreiter and Schmidhuber in 1997 as a way to address the vanishing gradient problem in traditional RNNs, which made it difficult to train deep networks. LSTMs achieve this by using a special type of memory cell that can store and update information over a longer period of time.

Since their introduction, LSTMs have been applied to a wide range of tasks, including language modeling, machine translation, and speech recognition. For example, in the field of natural language processing, LSTMs have been used to build language models that can generate coherent text, as well as to build translation systems that can translate between different languages. In the field of speech recognition, LSTMs have been used to build systems that can transcribe spoken language into text.

Ultimately, LSTM is a fitting architecture for our target task. There has been a significant amount of work on using LSTMs in combination with convolutional neural networks as well, as other types of neural networks, to improve performance on a wide range of tasks. Therefore, we found it fitting to use LSTMs to create temporal features between the successive CNN feature sets.

C. Event Based Camera Tracking

In recent years, researchers have proposed robust object tracking methods for event-based cameras or dynamic vision sensors (DVS). These sensors can be challenging to work with due to noise events, rapid changes in the event stream, complex background textures, and occlusion. To address these challenges, several authors have used a correlation filter mechanisms and convolutional neural network (CNN) representation. Similar to our approach, the appearance of event objects are represented using features from the hierarchical convolutional layers of a pre-trained CNN - often leveraging Feature Pyramid Networks (FPNs). The results is that researchers are able to achieve good tracking performance in complicated scenes with noise events, complex backgrounds (such as from high-speed in autonomous driving environments), and other occlusions.

IV. Approach

A. Dataset

We used the Event-based Vehicle Detection and Tracking Dataset[3] from the University of Michigan's Rawashdeh Research Group to train our network. The inputs to this 1.11GB dataset are the data streams from the event camera, while the ground truths are the bounding boxes. These bounding boxes are labelled at a constant frequency of 24, 48, 96, 192, or 384 Hz. The event camera used to collect the data was the DAVIS 240c.

The data was collected at two different scenes at the campus of the University of Michigan-Dearborn. In both scenes, the event camera is placed on the edge of a building while pointing downwards at the street, representing an infrastructure camera setting. No ego-motion is applied to the camera (the camera is static). A

sample of each scene's view using the grayscale camera is shown below.

Figure 1: Sample images from Event-based Vehicle Detection and Tracking Dataset

B. Preprocessing

This dataset was particularly difficult to work with. The dataset was primarily tabular data mixed with grayscale images. We didn't use the grayscale images for our purposes, but rather the event camera data. The data for each event captured by the event-based camera was a timestamp in microseconds, an x-y coordinate w.r.t. a standard digital image reference frame, and polarity - which measures if the intensity of a pixel increases or decreases.

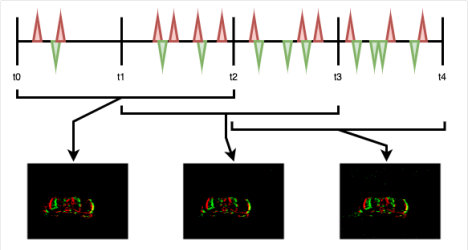


Figure 2: Event Camera Data Bucketing

In order to analyze the event data collected from a 24 Hz camera, we divided the intervals between each image into four smaller intervals, labeled t_0 through t_4. We then created three buckets of events, with each bucket representing approximately half of the total interval. Bucket 1 (B1) consists of events from t_0 to t_2, Bucket 2 (B2) consists of events from t_1 to t_3, and Bucket 3 (B3) consists of events from t_2 to t_4. These buckets allowed us to bucket data that occurred and may have otherwise been lost without event based data. To represent the events within the buckets, we generated image frames with dimensions $(W, H) = (240, 180)$ and two channels: one for increasing intensity polarities and one for decreasing intensity polarities. To simplify the analysis, we initially filtered out data with more than one bounding box. To efficiently load the dataset and ground truth data, we developed a method to load the file paths and another method to load the datasets into a dataframe. Using the timestamps provided in the event camera data, we were able to aggregate the events into the aforementioned buckets and return the final event frames.

C. Network Architecture

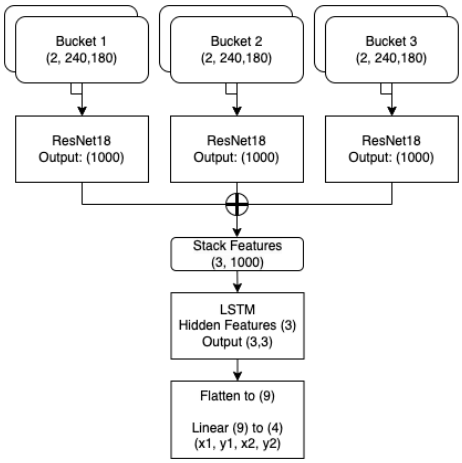


Figure 3: Architecture Diagram

Our model needed to not only extract features from sparse information image frames, but also capture the temporal features from the sequential frames. Ultimately, our architecture needs to output four values corresponding to the bounding box of the object we are aiming to track/detect. Bounding box detection involves identifying the location and extent of objects in an image.

A CNN architecture would allow us to extract features from the frames. We used a the ResNet18 CNN model [4] as the backbone for our feature extraction because ResNet has a great performance for feature extraction for classification problems, and ResNet18 is a lightweight and effective model architecture.

We used LSTM [5] as our RNN architecture for temporal features. Each of the three buckets of event frames is fed through a feature extractor, which returns a feature vector of size 1000 for each bucket. The LSTM takes in the CNN feature vectors from each bucket as the input, where the vector is the input and the hidden size is dictated by the number of buckets.

Lastly, the flattened output of the LSTM gets fed into a Linear Layer that predicts the bounding box from the features of the LSTM. This is length 4 tensor with the values (x_1, y_1,x_2,y_2) corresponding to the predicted bounding box.

D. Training

1) Optimizer: We used the Adam optimizer [6] which is an stochastic optimization method that is widely used for training. It combines the benefits of the Adagrad

and RMSprop optimization algorithms, and adds the concept of momentum to accelerate the convergence of the optimizer. One key feature of Adam is its ability to adaptively tune the learning rates for each weight in the model, which helps to prevent oscillations and improve the speed of convergence.

2) Loss Functions: We experimented with a couple of different loss functions.

The first loss function used was Intersection Over Union (IoU), also called the Jaccard Distance, $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Notice that $0 \leq J(A,B) \leq 1$ with 0 being for disjoint sets and 1 being for equivalent sets. When applied to the bounding boxes, it is the ratio of the intersection area to the union of the area of the two rectangles. In the optimal case, we want the predicted box and the ground truth to be exactly the same, $A \cap B = A \cup B = 1 \implies J(A,B) = 1$. Since we generally want a loss function to minimize, it is common to use $\mathcal{L}(\hat{y}, y) = 1 - J(\hat{y}, y)$. Also notice that $J(A,B) = 0$ whenever the two bounding boxes do not overlap, but there is no way to penalize the concept of being "further apart" or "more wrong" when the boxes do not intersect at all. We use a modified variant, Distance IoU Loss[7], that applies an extra penalty when the center of the two boxes is not aligned to account for this, $\mathcal{L}(\hat{y}, y) = 1 - J(\hat{y}, y) + \mathcal{R}(\hat{y}, y)$.

The other loss function we tried was Mean Squared Error (MSE) Loss. $\mathcal{L}(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$. It is defined as the average of the squared differences between the predicted output and the true output. MSE loss is commonly employed in tasks such as regression and classification due to its ease of computation and interpretability. One key advantage of using MSE loss is that it penalizes large errors more heavily than small errors, which helps to stabilize the model and prevent overfitting. This is achieved through the use of the squared term in the MSE loss function. Additionally, MSE loss is differentiable, allowing it to be utilized with optimization algorithms such as gradient descent to adjust the model's parameters and improve its performance.

V. Experimental Results

The main objective behind our architecture is to output the four dimensions of the bounding box as a prediction, this is called Bounding Box Regression. Bounding-box regression is a popular technique in object detection algorithms used to predict target objects' location using rectangular bounding boxes. It aims to refine the location of a predicted bounding box. Bounding box regression uses the overlap area between the predicted bounding box and the ground truth bounding box referred to as Intersection over Union (IOU) based losses.

The IoU loss is naturally better for bounding box regression as compared to MSE loss and hence we can see that the train loss for IoU loss converges much quicker as compared to MSE loss for the same application.

We notice that the training and validation curves do not have any unreasonable differences between them so that would ensure that our model is not over-fitting on the training data. We also notice that we have a reasonable validation loss considering the data was sparse in certain situations resulting in sub-optimal features.

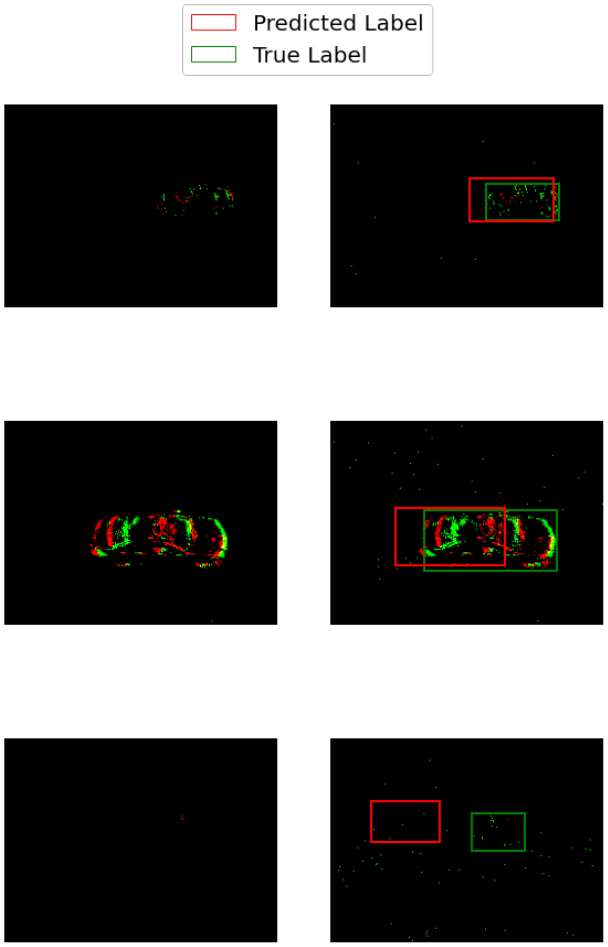


Figure 6: Results Visualized: Dense vs. Sparse Data

The visuals depicted in Figure 4 and Figure 5 (refer to references) indicate that the model demonstrated an improvement in both training and validation error over the number of epochs when utilizing both Intersection over Union and Mean Squared Error loss functions.

However, as depicted in Figure 6, the quality of the bounding box exhibited significant variation based on the density of data within the bucket. In the case of highly dense data, as seen in the images of Case 1 and Case 2, the model generated relatively accurate bounding boxes. Conversely, when the data was sparse, as seen in

Case 3, the model struggled to extract meaningful features and produced significantly less accurate bounding boxes. Hence the quality in the bounding box is heavily depending on the sparsity of the data in the bucket.

VI. Discussion

One possible adjustment to the model that could be made would be to have a variable time length for the bucket to ensure that there is sufficient features in the data to make a meaningful bounding box prediction. We could have also used other functions such as Batch Normalization or activation layers to extract the most amount of features from the non-linear and non-standardized data.

Fusing Frame Camera data: Since Event Based Cameras only detect changes in intensity $\frac{\partial \text{Scene}}{\partial t}$, they have no way of knowing the whole scene. $\int \frac{\partial \text{Scene}}{\partial t}\, dt = \text{Scene} + C$. For some notion of the start time to the current time, we have

$$\int_{t_{start}}^{t_{curr}} \frac{\partial \text{Scene}}{\partial t}\, dt = \text{Scene}(t_{curr}) - \text{Scene}(t_{start})$$

which is not enough to determine the current scene, unless one starts from the beginning of time. For example, imagine a car that drives into the scene before the starting time and then stays parked. Unless the task does not require knowing the current scene, it would be beneficial to fuse Event Based Camera data with normal camera data which gives us $\text{Scene}(t_{start})$ as a baseline.

Ego-motion: Many potential applications for Event Cameras involve ego-motion such as on autonomous vehicles or drones. Rather than a fixed camera detecting movement in the scene, the camera will be moving with the object, and hence will be able to produce much denser data from the neurons due to the sensors facing a different area regularly. Hence, combined with gyroscopic and accelerometer data, this has the potential to be much more accurate in predicting not only bounding boxes but also classification and object location prediction.

Stereo Cameras/ Depth Estimation: Being able to estimate the depths of objects adds an entire new dimension to the data. Possible methods to implement this could include stereoscopic cameras or incorporating optical flow, ego-motion, and visual inertial odometry.

Classification: Being able to differentiate between trees, the road, and pedestrians would be invaluable for autonomous cars. The low latency of event cameras would be able to detect and react faster to obstacles and thus improving overall safety.

Working with the event stream: One possible approach we did not take was to deal with the event stream directly without grouping the data into frames. This would require a different approach, but might allow for even lower latency. This can be done using a variety of techniques, such as optical flow, structure from motion, or inverse depth. An additional challenge is that this would necessitate periods of incredibly sparse data and periods where the event flux is extremely high.

References

[1] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, jan 2022.

[2] Eric D Manley, Huzaifa Al Nahas, and Jitender S Deogun. Localization and tracking in sensor systems. In *IEEE International conference on sensor networks, ubiquitous, and trustworthy computing (SUTC'06)*, volume 2, pages 237–242. IEEE, 2006.

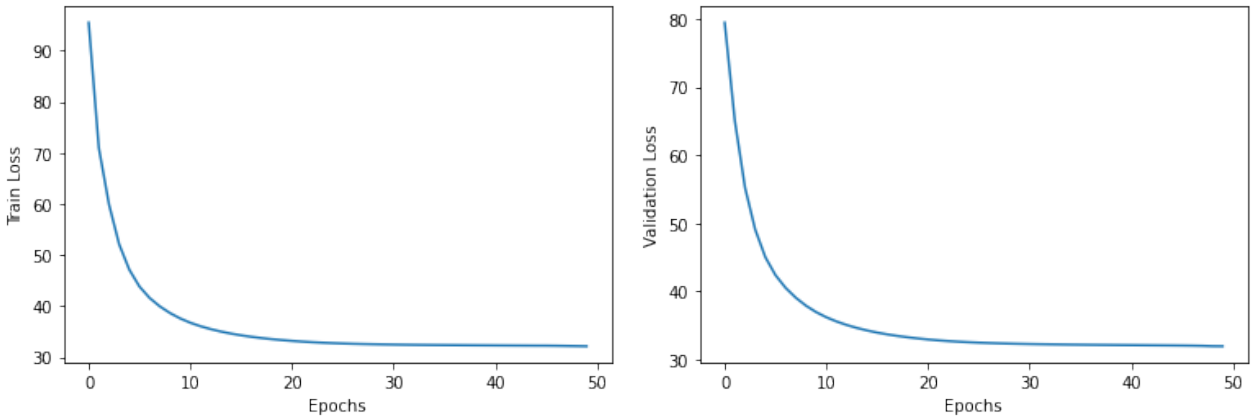
[3] Zaid El Shair and Samir A Rawashdeh. High-temporal-resolution object detection and tracking using images and events. *Journal of Imaging*, 8(8):210, 2022.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

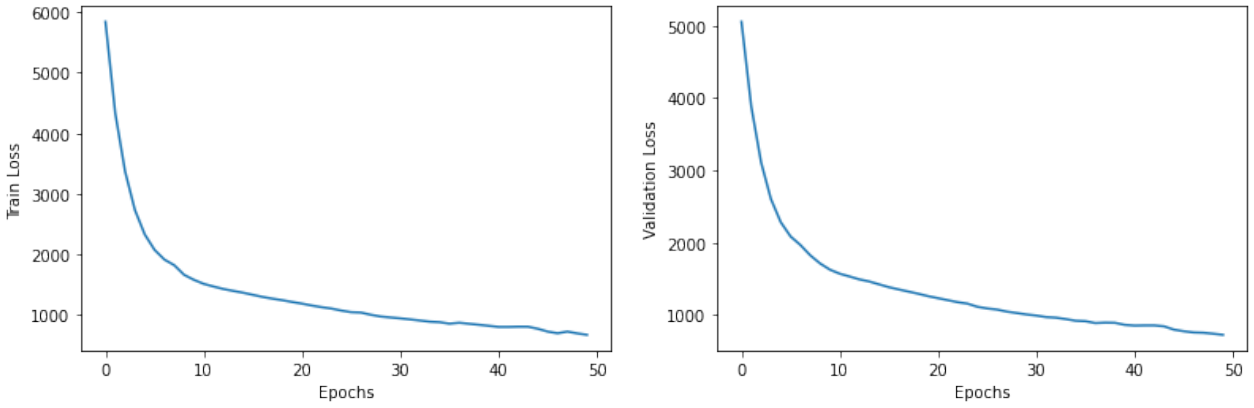
[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[7] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression, 2019.



(a) Train (b) Validation

Figure 4: Intersection over Union (IoU) Loss vs. Epochs



(a) Train (b) Validation

Figure 5: Mean Squared Error (MSE) Loss