EMOD: Efficient Moving Object Detection via Image Eccentricity Analysis and Sparse Neural Networks

Xiaomin Li[1,2], Ramin Nabati [2], Kunjan Singh [2], Enrique Corona [2], Vangelis Metsis [1] Armin Parchami [2]
[1]Texas State University, [2]Ford Motor Company

Abstract

This paper proposes an efficient moving objects detection pipeline focusing on dynamic object detection on video streams captured by traffic monitoring cameras. While developing autonomous vehicle systems, we found that views from self-driving vehicles can be occluded by dynamic or static objects on the street. Whereas infrastructure nodes such as traffic monitoring cameras having broader field-of-views and better perspectives can be used as auxiliary sensors to share traffic information with nearby self-driving cars in real-time. However, these infrastructure cameras usually have constrained computation resources, and detecting hundreds of static background objects in consecutive video frames is wasteful. In our detection pipeline, we leverage the image eccentricity analysis as a pre-processing step to fast generate moving objects segmentation maps. These maps are used to mask the original images to get images that only contain the moving objects in the scene. These sparse images are then passed to an object detection model built with a sparse convolution backbone network, resulting in significant reduction in computational costs. Our quantitative experiments illustrate that the proposed detection pipeline can achieve up to 50% inference speedup with negligible detection accuracy drop in images obtained from traffic monitoring cameras.

1. Introduction

Developing self-driving cars has become one of the hottest machine learning research fields in recent years. A fully automated self-driving car system will revolutionize daily human transportation and create a magnificent business impact. With the success of self-driving cars, smart city infrastructure will also become vital. An intelligent traffic management system mounted at higher vantage points provides a broader coverage compared to a self-driving car. Such systems can be used to monitor and control traffic, alarm nearby vehicles and share information among vehicles.

Traffic monitoring cameras are the most commonly used sensors that are installed on street infrastructure nodes. Since they have broader coverage than the cameras installed on self-driving cars, they can share what they see with multiple nearby self-driving cars to prevent accidents caused by obstructed views of these vehicles. We can use these cameras to conduct object detection tasks and alert anomalies to upcoming self-driving cars. However, running real-time object detection tasks on these edge devices requires significant computational resources. Due to data privacy constraints and data transmission overheads, most of these computations need to be done directly on edge devices and can't be offloaded to a cloud device. To achieve real-time response and the best possible performance, we must carefully arrange the computation resources and design the algorithms efficiency to meet the real-time response requirements in autonomous driving application.

An ordinary object detection model selects thousands of multi-scale image segments and applies an image classification task to each segment. This is a very compute-intensive task for edge devices such as infrastructure cameras. In this project, we introduce an efficient object detection pipeline used for traffic monitoring systems which significantly reduces the required resources and decreases inference time. We observe that consecutive frames captured by static cameras share many similarities in background and static objects. It is unnecessary to conduct object detection pixel by pixel and frame by frame with such images. Therefore, we propose a background and stationary object subtraction method to pre-process the input images to filter our the static and background areas, and only apply the detection algorithm on areas with dynamic objects.

The Finite Memory Eccentricity algorithm introduced in [4] is a non-parametric, assumption-free methodology for extracting information from data. It can be computed recursively, and requires very limited statistical information when processing input images. We leverage this algorithm to generate moving object segmentation maps as a pre-processing step to get images that only contain the difference among consecutive video frames. Such a method only adds negligible computation overhead when pre-processing the input images.

The sparse convolution networks take data lists and index lists as the input image format and only apply convolution computation on these nonempty pixels. To incorporate the sparsity in the pre-processed images and improve inference time, we use a sparse convolution library [10] to build object detection models. The generated segmentation maps generated by the eccentricity algorithm are used as masks on the original images, so the resulting sparse images only contain moving objects. The sparse images will be stored in a compressed format that has lists of pixel values and corresponding indexes. The sparse convolution-based object detection model takes such inputs and predicts objects the same way as ordinary object detection models.

We conduct qualitative and quantitative experiments to test the performance of our sparse convolution-based object detection model. The experiments show that on a single CPU, the sparse object detection model on average achieves 50% inference speedup compared to the dense counterpart (standard object detection model). We also leverage the image eccentricity algorithm to generate moving object segmentation maps and test the performance of the object detection model on the dataset collected from real-world traffic monitoring cameras. The experimental results show that our model can achieve very similar detection accuracy but, at the same time, significantly reduce the computation amount needed for a model running on edge devices.

To summarize, the contributions of our proposed pipeline are as follows:

- We utilized the image eccentricity analysis algorithm to develop a fast moving object extraction method from consecutive video frames with a negligible amount of computation overhead.
- We leveraged sparse convolution networks to build object detection models that only apply convolution computations on nonempty pixels and largely reduce the needed inference computation compared to standard object detection models.
- We qualitatively and quantitatively analyzed the performance of our sparse object detection models on multiple datasets. The experiment results showed that our detection pipeline could speed up inference time by 50% with little to no detection accuracy drop.
- We explored the relationships among the quality of generated segmentation maps with different eccentricity analysis algorithm thresholds.

2. Related Work

2.1. Efficient object detection

Object detection models are mostly using image classification models as backbones. However, a one-time object detection process often divides the input image into multiple small regions. It conducts image classification on each region, which results in the object detection task requiring hundreds or thousands more computations than an image classification task. To achieve real-time object detection, researchers have developed multiple strategies to reduce the required computation while trying to retain the detection accuracy. One approach was to design lightweight backbone networks such as the EfficientDet [31] backbone networks and YOLO families [17]. Also, the efficient deep learning model ideas can be used to reduce object detection computations, such as network pruning [8, 14], network quantization [5, 16], and model-hardware co-design [27, 22, 24]. Another strategy is to define a multi-scale detection strategy and only spend more computation resources on the area of interest or hard-to-detect small objects [23, 29, 25].

Another aspect of efficient object detection is preprocessing the input images to reduce the total regions that need to apply object detection. For example, salient object detection is a task based on a visual attention mechanism in which algorithms aim to explore objects or regions more attentive than the surrounding areas in the scene. Given original images and segmentation maps as ground truths, the model tries to predict the salient objects in the images [13, 32]. When processing videos instead of single images, optical flow and feature warping techniques can be used together to depict similar regions among consecutive frames and reduce redundant computations on such regions without affecting detection accuracy [18, 20].

The eccentricity analysis algorithm used in our work is similar to optical flow as it captures image changes among consecutive video frames. However, it gives more detailed spatial information than optical flow [4]. Our goal is to only detect moving objects on the videos collected from static cameras. To prove the efficacy of our proposed pipeline, we used relatively simple object detection models and detected objects from videos frame by frame. But our pipeline is also compatible with the earlier mentioned efficient object detection methods.

2.2. Sparse Convolution

Convolution networks are the de-facto standard for analyzing spatio-temporal data such as images, videos, and 3D shapes. Standard "dense" implementations of convolution networks are inefficient when applied to sparse data, such as 1D lines on RGB images and 3D point clouds on RGB-D images. Sparse Convolution plays a vital role in processing such kinds of data, unlike the regular convolution computations that store image information as matrix or tensor and process them using dense matrix multiplication. The sparse images can be represented as data lists and index lists, and the sparse convolution uses a particular schema to process such a data format. That is, the sparse convo-

lution collects all convolution kernel computations (atomic operations) and saves them in a Rulebook as instructions for computation. It is more efficient because we do not need to scan all the pixels in an image but only calculate the convolutions for the nonzero elements. The sparse convolutional networks were first studied in the works [6, 9, 26]. Their major drawback is that the active region proliferates with each layer, and the sparsity of feature maps from each layer will decrease. The work [12] designed a submanifold sparse convolution network which restricts the growth of active regions. It performs on par with state-of-the-art methods whilst requiring substantially less computation. Since then, sparse convolution has been used for 3D point cloud processing [6, 30, 15]. Our work utilized the sparse convolution library developed from [12] to build the object detection model and process the sparse 2D images generated by the eccentricity analysis algorithm.

3. Methodology

3.1. Processing Pipeline

The proposed efficient object detection process is depicted in Figure 1. The consecutive video frames are the original inputs. The image eccentricity analysis algorithm will pre-process these images to generate segmentation maps containing only moving objects. Then the segmentation maps will be masked on original inputs to get moving objects' foreground images. The generated foreground images are in a compressed format containing only nonempty pixel values and their corresponding indexes. After that, the foreground image will pass through a sparse convolution-based object detection model to detect the objects contained in the foreground images. In this pipeline, The image eccentricity analysis is a purely mathematical method that iteratively updates the current frame segmentation map, which only involves a small amount of computational overhead. Meanwhile, the sparse convolution object detection model significantly reduces object detection inference computations than commonly used dense object detection models.

3.2. Image Eccentricity Analysis

Eccentricity analysis is a method to extract information from time-series data without predetermined parameters, random variables, or distributions. This methodology was first introduced in the work [1]. Thereafter, such a method has been implemented on various tasks, such as clustering [3], classification [19], and fault detection [2].

The Eccentricity ζ of the data sample x at the time instant k can be defined as [1]:

$$\zeta_k = 2\frac{\sum_{i=1}^{k} d(x, x_i))}{\sum_{i=1}^{k}\sum_{j=1}^{k} d(x_i, x_j))} \tag{1}$$

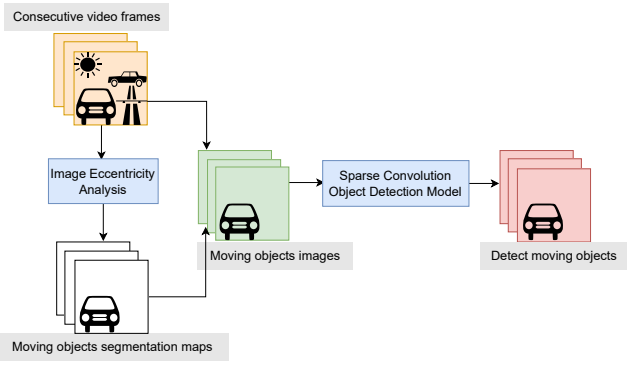


Figure 1: EMOD:Efficient moving object detection pipeline.

$$k \geq 2, \sum_{i=1}^{k}\sum_{j=1}^{k} d(x_i, x_j) > 0$$

where d is some type of distance, such as Euclidean distance, Mahalanobis distance, etc. Eccentricity is bounded by [1]:

$$0 \leq \zeta_k \leq 1, \sum_{i=1}^{k} \zeta_k = 2$$

When d is Euclidean distance, the equation 1 can be derived as [1]:

$$\zeta_k = \frac{1}{k} + \frac{(\mu_k - x_k)^T(\mu_k - x_k)}{k\sigma_k^2} \tag{2}$$

$$k \geq 2, \sigma_k^2 > 0$$

Both the mean μ_k and the variance σ_k^2 can be recursively updated, respectively, by [1]:

$$\mu_k = \frac{k-1}{k}\mu_{k-1} + \frac{x_k}{k} \tag{3}$$

$$k \geq 1, \mu_1 = x_1$$

$$\sigma_k^2 = \frac{k-1}{k}\sigma_{k-1}^2 + \frac{(\mu_k - x_k)^T(\mu_k - x_k)}{k-1} \tag{4}$$

$$\sigma_1^2 = 0$$

The work [4] addressed the infinite memory problem involved in the above method. With equations 1&2, the Eccentricity computation takes into account all data samples going back to $k = 1$ frames, which can be a significant problem in dynamic and rapidly evolving environments. Therefore, this work introduces a finite memory mechanism for updating Eccentricity where it exponentially forgets older samples during the recursive computation.

A constant forgetting factor $\alpha = 1/k, 0 \leq \alpha \leq 1$ is introduced to assign a set of exponentially decaying weights to the older data sample x_k [4], hence the equation 2 rewrites as:

$$\zeta_k = \alpha + \frac{\alpha(\mu_k - x_k)^T(\mu_k - x_k)}{\sigma_k^2} \tag{5}$$

The effect of equation 5 will cause the older data points to essentially be eliminated from the eccentricity computations after a certain data point x_k.

A variance threshold γ is chosen to prevent $\sigma_k^2 \approx 0$ when consecutive samples are very close to each other [4]. Therefore, the Eccentricity formula becomes:

$$\zeta_k = \alpha + \frac{\alpha(\mu_k - x_k)^T(\mu_k - x_k)}{max(\sigma_k^2, \gamma)} \tag{6}$$

$$\gamma = \frac{\alpha * (1 - E\_sen)^2}{1 - \alpha}$$

The threshold γ is partially dependent on α, and we can tune the parameter E_sen to get different γ values.

The normalized ζ_k in equation 6 written as:

$$\varepsilon_k = \frac{\zeta_k - \alpha}{1 - \alpha} = \frac{\alpha(\mu_k - x_k)^T(\mu_k - x_k)}{(1 - \alpha)max(\sigma_k^2, \gamma)} \tag{7}$$

3.3. Moving objects segmentation from Image Eccentricity

The finite memory eccentricity formulation can be adapted to image streams. For example, a 3-channel RGB image at time k can be represented as $x_k = \left\{R_k^{i,j}, G_k^{i,j}, B_k^{i,j}\right\}$, where (i, j) is a pixel index of the image. For an image in a video stream, each pixel is independent from the others and we treat it as a separate data stream. Therefore, each pixel's eccentricity value is computed based on equation 7. At each time instance k, an eccentricity map, E_k, with the same size of the image will be generated [4].

We can generate a moving objects segmentation mask S_k based on the eccentricity map E_k with an anomaly detection threshold M [4], where

$$S_k = \begin{cases} 0, & E_k \leq M \\ 1, & E_k > M \end{cases} \tag{8}$$

For a pixel location, if its ε_k value is smaller than the threshold M, we can consider that at the current video frame, this pixel value does not significantly change compared with previous frames. So it can be considered stationary and assign a 0 value in the segmentation map. On the contrary, whenever the ε_k is greater than the threshold M, this pixel is considered part of a moving object, so assign 1 value in the segmentation map.

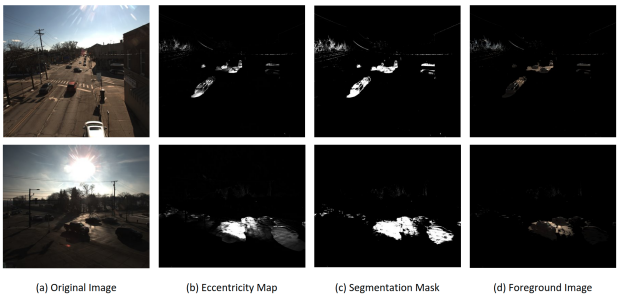


Figure 2: Visual examples of Eccentricity-based images. (a) are video frames captured by traffic monitoring cameras mounted on vantage points. (b) are the eccentricity maps generated based on previous frames. (c) are moving object segmentation maps that are generated based on equation 8. (d) are sparse foreground images that only contain moving objects pixel values.

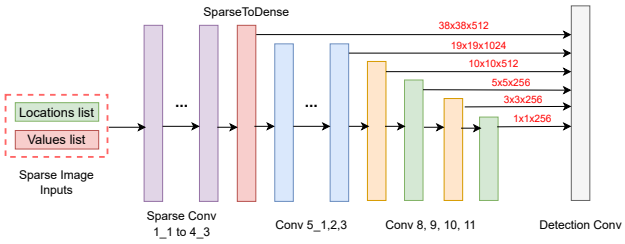


Figure 3: Sparse convolution based single shot object detection (Sparse-SSD) model.

The threshold M is defined as:

$$M = \frac{\alpha(m^2 + 1)}{2}$$

We use $m = 3$ in all experiments. Some examples of E_k and S_k are shown in Figure 2.

3.4. Sparse Convolution object detection models

We modify a VGG-based [28] single shot object detection (SSD) model to use sparse convolution instead of reagular dense convolution. The model architecture is shown in Figure 3. This model takes the compressed format sparse images as input, which contains a list of pixel indices of nonempty pixels and a list of corresponding pixel values. When building the Sparse-SSD model, we replace the first ten convolution layers (from 1_1 to 4_3) of the VGG backbone network with sparse convolution layers using the deep learning model library introduced in [11]. Then the output of the convolution 4_3 layer will be passed through a SparseToDense layer to convert the feature map to the dense format. After that, the model architecture is the same as the original VGG16-SSD model design [21].

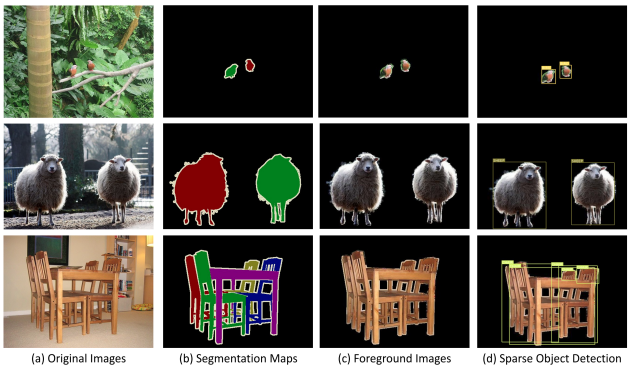


Figure 4: Pascal Dataset foreground image processing and detection examples. (c) Foreground images are generated by applying (b) segmentation maps to (a) Original images. (c) images will be transformed to the compressed format and passed to the Sparse-SSD model 3. (d) shows the detection output of inputs (c). The Sparse-SSD model has a similar detection performance to the Dense counterpart.

4. Experiment and Results

4.1. Sparse model inference speedup

The first experiment is to reveal how sparse convolution-based object detection models can help decrease the inference time.

4.1.1 Data pre-processing

The PASCAL [7] dataset is selected to conduct this experiment. We use the PASCAL 2007 and 2012 trainval set as our model training data and the 2007 test set as testing data. Such data split is consistent with the original SSD paper [21]. The model is trained with original full images. When preparing the foreground objects sparse images, we use the provided segmentation maps from the dataset as masks and apply them to the original images. The generated foreground images are used as inputs to the model for inference. Figure 4 shows a few processing and detection examples.

4.1.2 Model training and testing setup

We train a dense version SSD300 model on the training set described in section 4.1.1. The model is trained on a single GPU, with a starter learning rate 1e-3. The model is trained for 120000 iterations with the learning rate decaying by 90% at the iterations 8000 and 10000. The momentum is set to 0.9 and weight decay set to 5e-4. The trained model testing on dense testing set mAP is **0.759**. Such result is correspond with the original paper [21].

	Mean Average Precision	
Data Split	Dense Model	Sparse Model
All Foregrounds	0.747	0.739
0%_50% Sparsity	0.855	0.829
50%_60% Sparsity	0.770	0.858
60%_70% Sparsity	0.837	0.846
70%_80% Sparsity	0.821	0.792
80%_90% Sparsity	0.677	0.666
90%_100% Sparsity	0.65	0.79

Table 1: The Dense-SSD and Sparse-SSD detection accuracy on different subsets of the foreground PASCAL test set.

4.1.3 Load pre-trained weights to Sparse-SSD model

The Sparse-SSD model parameters we used for experiments are loaded from the pre-trained Dense-SSD model. However, the Sparse-SSD model built based on the sparse convolution library [11] has a different saved model parameter shape. We will have to convert the dense model parameters from each layer to the proper shape that fits the Sparse-SSD model. We first get each layer's state_dict from the Dense-SSD. Suppose a layer is a convolution layer, and the corresponding layer in the Sparse-SSD is a sparse convolution layer. In that case, we reshape such state_dict from the shape $[output, input, kernelwidth, kernelheight]$ to the shape $[kernelwidth * kernelheight, 1, input, output]$. For example, a convolution layer has the weight tensor shaped $[128, 64, 3, 3]$. After conversion, its shape will become $[9, 1, 64, 128]$. Then we apply the converted convolution weight matrix to the sparse convolution. If the current layer is not a convolution layer, such as bias or pooling, we apply the same state_dict from a Dense-SSD layer to a Sparse-SSD layer.

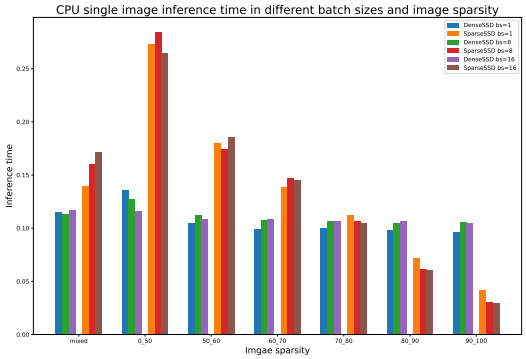
4.1.4 Inference performance comparison

At first, we verify if the Sparse-SSD model can successfully detect objects from sparse inputs and how it performs compared with the Dense-SSD model. We prepared foreground test images based on the PACAL2007 test set and separated them into groups by different sparsity ratios. An image's sparsity ratio is computed as:

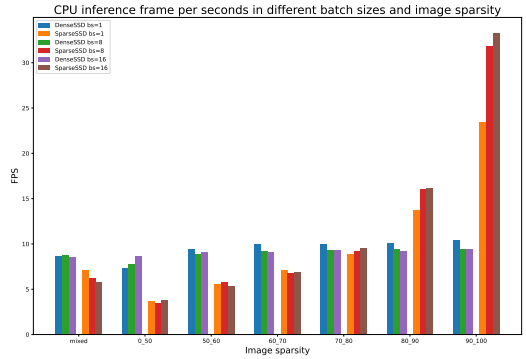
$$sparsity = number\ of\ empty\ pixels / image\ size \quad (9)$$

From the table1, we can see that the Sparse-SSD model performed as well as the Dense-SSD model in all sparse image groups. Such results verify the efficacy of using Sparse-SSD to detect objects from sparse images.

In the next experiment, we want to know how the Sparse-SSD model can speed up the object detection inference process. Figure 5a and Figure 5b show our findings. We



(a) The average single image inference time on PASCAL foreground images.



(b) SSD model inference FPS on single CPU.

Figure 5: Inference speed comparison between Sparse-SSD and Dense-SSD when input images have different sparsity ratios.

compute the single image average inference time and FPS (frames per second) on Dense-SSD and Sparse-SSD. The testing set is the same as the ones used in Table 1. From the experimental results, we can find that the inference speed of the Dense-SSD model is stable with various sparse ratio inputs. However, the Sparse-SSD inference speed largely depends on how sparse the input images are. The Sparse-SSD model we build will achieve significant inference speedup when the input images are more than 80% sparsity. When the inputs have more than 90% sparsity ratio, we can achieve nearly 2x speedup compared to Dense-SSD inferences.

4.2. Image eccentricity analysis speedup

From section 4.1.4, we verified that when foreground images are generated with accurate segmentation maps, the Sparse-SSD model can achieve equivalent detection performance compared to the Dense-SSD model. And when the input images are sparse enough, the Sparse-SSD model can achieve significant inference speedup. Therefore, in this experiment, we will test how the eccentricity algorithm-based segmentation maps can help to extract moving foreground objects and how the proposed processing pipeline can be used in real-world scenarios.

4.2.1 Dataset and data pre-processing

We used a real-world surveillance dataset collected by the Ford research group to test the performance of the proposed method. This dataset is collected from stationary cameras at different street intersections. It contains a total of 1081 recordings which were collected at different locations, under different weather and lighting conditions, and across a half-year period. Two data examples are shown in Figure 2(a). Each record has around 2-3 minutes consecutive frames with a 10FPS collection rate. We divide the total recordings into an 80/20 split for the training set and testing set, resulting in 1,380,320 images in the training set and 345,132 images in the testing set. Since the images in a recording are mostly identical, to reduce the training time, we shuffled the images in the training set and randomly selected 10% of the images to train the model. Furthermore, since the eccentricity analysis will filter out any stable objects, we need to modify the annotations to only retain moving objects. There were 18 different object categories labeled, but we only selected 6 of them during training and testing. The categories retained were Bus, Car, Pedestrian, Pedestrian With Object, Truck, and Construction Vehicle. We also marked very small objects and partially occluded objects as difficult to detect objects.

4.2.2 Experiment setups and results

We use the same VGG16-SSD300 architecture as used in section 4.1. The model is trained on Google Cloud compute machine with 2 NVIDIA Tesla T4 GPUs. The training hyper-parameters are set the same as in section 4.1 except for training it with more iterations. The trained model testing mAP is **0.215**. The average precision for each categories are Bus: 0.276, Car: 0.296, Construction Vehicle: 0.180, Pedestrian: 0.137, Pedestrian With Object: 0.125, Truck: 0.276. Such detection results are satisfied considering the detection difficulty of this dataset.

We select a single recording to test the Sparse-SSD inference speedup with the help of eccentricity analysis generated segmentation maps. This recording contains a total number of 2664 images. The distribution of foreground image sparsities are shown in Figure 6. Such images are generated with eccentricity threshold $\alpha = 0.1$ and $E\_sen = 0.3$. We can observe from Figure 6 that almost all generated foreground images have a sparsity ratio of more than 90%.

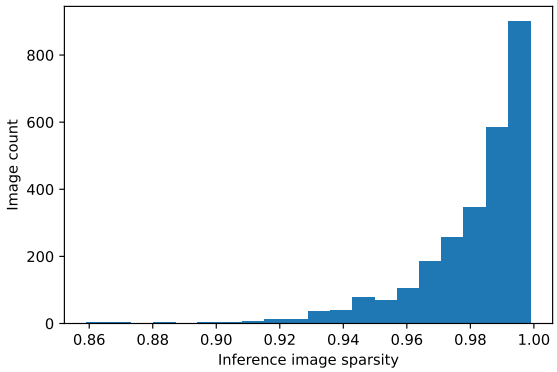


Figure 6: Foreground images sparsity ratio count in a 4.5 minutes traffic camera captured video segment.

	Dense-SSD	Sparse-SSD
Total Inf time (s)	309.227 ± 7.220	190.082 ± 1.417
Single Inf time (s)	0.115 ± 0.022	0.072 ± 0.016
FPS	8.61	14.01

Table 2: The inference time comparison between Dense-SSD and Sparse-SSD on real-world surveillance dataset.

Recall from the results we got from section 4.1.4, within this sparsity ratio, the Sparse-SSD model can significantly improve the inference speed. Table 2 illustrate such findings in detail.

4.3. Eccentricity thresholds ablation study

This experiment studies how different eccentricity thresholds affect the foreground images' sparsity and inference speed. Two eccentricity thresholds can be tuned to generate different sparsity-level foreground images. One is a constant forgetting factor, $\alpha \in [0, 1]$. This parameter is the reciprocal of the moving k (how many previous frames to consider to compute the current frame eccentricity map). The smaller the α, the more frames to consider, the more eccentricity computations, and the more detailed foreground images. The other one is eccentricity sensitivity threshold, $E_{sen} \in [0, 1]$. The smaller the E_{sen}, the less sensitive the current eccentricity map is to eccentricity value changes, and the sparser the foreground images are. A few examples of different thresholds generated segmentation maps are shown in Figure 7.

We selected a combination of different α and E_{sen} values to test the average segmentation maps' sparsity and corresponding inference speed. The results are shown in Figures 8 and 9. These two figures show that inference time is inversely proportional to the input foreground image sparsity ratio. They also demonstrate how various eccentricity thresholds will affect image sparsity and inference speed.

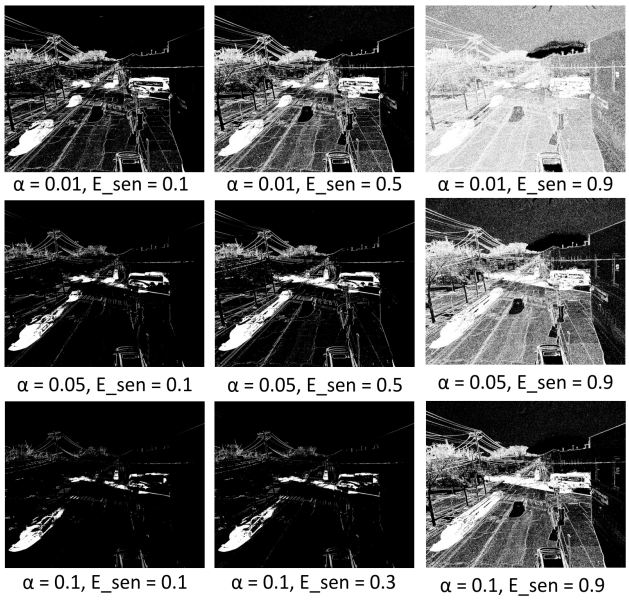


Figure 7: Examples of various Eccentricity thresholds generated segmentation maps.

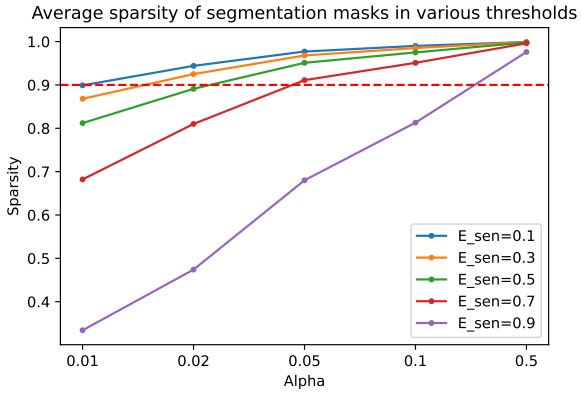


Figure 8: Eccentricity thresholds segmentation maps sparsity comparison with different combination of α and E_{sen}. The red horizontal line indicates the image sparsity from which we can see significant inference speedup using the Sparse-SSD model.

The red horizontal line in Figure 9 indicates the average inference time of the Dense-SSD model. Below this line, we find many combinations of eccentricity thresholds that can make Sparse-SSD achieve an inference speedup.

5. Discussion and future work

From Table 1 we find that the sparse-convolution will not cause detection performance degradation when the foreground objects are perfectly segmented. However, eccen-

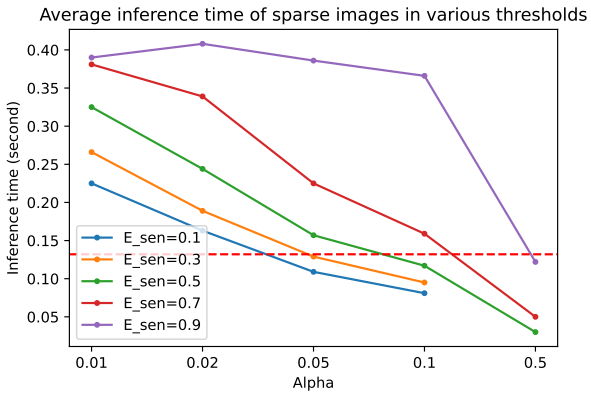


Figure 9: Sparse image inference time comparison with different combination of α and E_{sen}. The red horizontal line indicates the average inference time of the Dense-SSD model.

tricity algorithm generated segmentation maps will cause some foreground objects to be partially visible. This phenomenon will cause the detection accuracy to drop since the inference object features are distorted compared to the training samples. To solve this problem, we can consider adding another post-processing method to the segmentation maps to improve their quality. Meanwhile, we need to train the object detection model with sparse and noisy training samples instead of original images captured from cameras.

In future work, we will replace more dense convolution layers with sparse counterparts when building the object detection model. And create multiple sparse counterparts of start-of-the-art object detection models. The sparse convolution library we chose for experiments has not yet provided enough support to all kinds of convolution layers. They are also not built as effectively as they should compare with the latest features of deep learning libraries such as PyTorch, TensorFlow, etc. If we use a more efficient sparse convolution implementation, our detection pipeline could save significant overhead computation time and see more wall-clock time speedups.

6. Conclusion

This paper proposes an efficient moving object detection pipeline that utilizes the image eccentricity algorithm to generate moving object segmentation maps and leverage sparse convolutions to build object detection models to reduce inference-time computations when processing sparse images. We first studied the relationship between sparse-convolution-based object detection model inference speed and the sparsity ratio of sparse input images. Then we conducted experiments on a real-world traffic camera dataset to verify the efficacy of our detection pipeline. Among the eccentricity analysis algorithm, we studied how different eccentricity thresholds affect the quality of the generated segmentation maps, the object detection inference speed, and corresponding detection accuracy. The experimental results show that when the input images have a sparsity ratio of more than 80%, our prototype sparse-convolution object detection model can achieve a 1.5x - 2x inference speedup with negligible detection accuracy drop. Our experiments also find that in real-world static traffic cameras, the similarity among consecutive frames is more than 90%, which verifies that a huge amount of computations are redundant and can be safely removed from the object detection pipeline to speed it up. These results demonstrate the promising future of using traffic monitoring cameras as auxiliary sensors for autonomous vehicle systems.

References

[1] Plamen Angelov. Anomaly detection based on eccentricity analysis. In *2014 IEEE symposium on evolving and autonomous learning systems (EALS)*, pages 1–8. IEEE, 2014.

[2] Clauber Gomes Bezerra, Bruno Sielly Jales Costa, Luiz Affonso Guedes, and Plamen Parvanov Angelov. An evolving approach to unsupervised and real-time fault detection in industrial processes. *Expert systems with applications*, 63:134–144, 2016.

[3] Clauber Gomes Bezerra, Bruno Sielly Jales Costa, Luiz Affonso Guedes, and Plamen Parvanov Angelov. A new evolving clustering algorithm for online data streams. In *2016 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 162–168. IEEE, 2016.

[4] Bruno Costa, Enrique Corona, Mostafa Parchami, Gint Puskorius, and Dimitar Filev. Efficient data-driven encoding of scene motion using eccentricity. *arXiv preprint arXiv:2103.02743*, 2021.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[6] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361. IEEE, 2017.

[7] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] Ben Graham. Sparse 3d convolutional neural networks. *arXiv preprint arXiv:1505.02890*, 2015.

[10] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition, pages 9224–9232, 2018.

[11] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

[12] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[13] Ashish Kumar Gupta, Ayan Seal, Mukesh Prasad, and Pritee Khanna. Salient object detection techniques in computer vision—a survey. *Entropy*, 22(10):1174, 2020.

[14] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[15] Rui Huang, Wanyue Zhang, Abhijit Kundu, Caroline Pantofaru, David A Ross, Thomas Funkhouser, and Alireza Fathi. An lstm approach to temporal 3d object detection in lidar point clouds. In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[17] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[18] Zhengkai Jiang, Yu Liu, Ceyuan Yang, Jihao Liu, Peng Gao, Qian Zhang, Shiming Xiang, and Chunhong Pan. Learning where to focus for efficient video object detection. In *European conference on computer vision*, pages 18–34. Springer, 2020.

[19] Dmitry Kangin, Plamen Angelov, and José Antonio Iglesias. Autonomously evolving classifier tedaclass. *Information Sciences*, 366:1–11, 2016.

[20] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[22] Ivan Lobachev, Roman Maleryk, Svitlana Antoschuk, Denys Filiahin, and Mykhaylo Lobachev. Integration of neural networks into smart sensor networks. In *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pages 544–548. IEEE, 2018.

[23] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9745–9755, 2019.

[24] Nashwan Adnan Othman and Ilhan Aydin. A new deep learning application based on movidius ncs for embedded object detection and recognition. In *2018 2nd international symposium on multidisciplinary studies and innovative technologies (ISMSIT)*, pages 1–5. IEEE, 2018.

[25] George Plastiras, Christos Kyrkou, and Theocharis Theocharides. Efficient convnet-based object detection for unmanned aerial vehicles by selective tile processing. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, pages 1–6, 2018.

[26] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017.

[27] Yongming Shen, Michael Ferdman, and Peter Milder. Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 93–100. IEEE, 2017.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Bharat Singh, Mahyar Najibi, Abhishek Sharma, and Larry S Davis. Scale normalized image pyramids with autofocus for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3749–3766, 2021.

[30] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2530–2539, 2018.

[31] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[32] Wenguan Wang, Qiuxia Lai, Huazhu Fu, Jianbing Shen, Haibin Ling, and Ruigang Yang. Salient object detection in the deep learning era: An in-depth survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.