Space-Time Slicing: Visualizing Object Detector Performance in Driving Video Sequences

Lee, T.-Y.; Wittenburg, K.B.

TR2019-024 June 05, 2019

Abstract

Development of object detectors for video in applications such as autonomous driving requires an iterative training process with data that initially requires human labeling. Later stages of development require tuning a large set of parameters that may not have labeled data available. For each training iteration and parameter selection decision, insight is needed into object detector performance. This work presents a visualization method called Space-Time Slicing to assist a human developer in the development of object detectors for driving applications without requiring labeled data. Space-Time Slicing reveals patterns in the detection data that can suggest the presence of false positives and false negatives. It may be used to setsuch parameters as image pixel size in data preprocessing and confidence thresholds for object classifiers by comparing performance across different conditions.

IEEE Pacific Visualization Symposium (PacificVis)

Space-Time Slicing: Visualizing Object Detector Performance in Driving Video Sequences

Teng-Yok Lee[*] Kent Wittenburg [†]

Mitsubishi Electric Research Laboratories

Abstract

Development of object detectors for video in applications such as autonomous driving requires an iterative training process with data that initially requires human labeling. Later stages of development require tuning a large set of parameters that may not have labeled data available. For each training iteration and parameter selection decision, insight is needed into object detector performance. This work presents a visualization method called Space-Time Slicing to assist a human developer in the development of object detectors for driving applications without requiring labeled data. Space-Time Slicing reveals patterns in the detection data that can suggest the presence of false positives and false negatives. It may be used to set such parameters as image pixel size in data preprocessing and confidence thresholds for object classifiers by comparing performance across different conditions.

1 Introduction

Object detection is a computer vision technique needed for multiple applications, including video surveillance, robot navigation, infrastructure asset monitoring, and autonomous driving. Given an image, the goal of object detection is to locate image patches that contain objects, and classify the patches into object categories, such as vehicles and pedestrians in driving video frames. Rather than developing new object detection algorithms from scratch, R&D engineers often adapt existing object detection algorithms and work to achieve optimal accuracy in a new domain. Nowadays the implementations of many state-of-the-art object detectors are publicly available, including Fast RCNN [12], Faster RCNN [21], YOLO (You Only Look Once) [19], and SSD (Single Shot Detector) [17]. However, adapting these implementations to new domain applications is still tedious. The common development procedure may start with some labeled data, some new data, and one or more candidate object detection systems. In subsequent stages, the developers must select among candidate object detection algorithms given new data sequences, retrain the systems over multiple iterations, and choose among parameter settings such as image resolution and color normalization in preprocessing and confidence threshold settings in the last stage. If neural nets are being used, then there are decisions regarding the types and number of layers, learning rates, and the number of optimization steps. In all these development steps, the performance of the system must be evaluated and compared to alternatives, and labeled data is more often than not unavailable.

To overcome these challenges, this paper presents a visualization method called Space-Time Slicing, which is suitable for applications involving a fixed camera mounted on a vehicle moving on a relatively static ground plane. Using a novel 2D view of compressed space and time, this visualization reveals the significant relative motion of detected objects and provides cues about missing detections as well as comparative information across different testing runs. By linking with other more standard views, users can verify the cues that are made available through Space-Time Slices. For this purpose, we use a view of video frames with bounding boxes surrounding detected objects as well as a view of small multiples over a timeline to represent tracked objects. The following sections will cover related work, the details of our visualization method, use cases, and future work.

[*]e-mail: tlee@merl.com

[†]e-mail: kentwitt@gmail.com

2 Related Work

Prior work in video visualization has been used to interpret data from surveillance cameras, sports games [15], and medical imaging [14]. The survey by Borgo *et al.* [4] provides a comprehensive collection of video visualization techniques. A video can be summarized with a storyboard-like visualization [7, 13, 18] if the events can be defined and classified, or rendered as 3D spatiotemporal volumes [8, 10]. For videos with static backgrounds such as surveillance videos, the motion can be revealed via the motion trajectory of moving foreground objects [5] or even the pixel intensity over time [3, 22, 24]. However, the above-cited work mainly uses object detection and other computer vision algorithms to generate visualizations to overview video content. The methods are not designed to facilitate the task of reviewing object detector performance. The data we are most concerned with, such as driving video sequences, involves moving cameras and backgrounds that frequently change. Our focus is on correctly detecting objects in such scenarios, not on summarizing or characterizing the behavior of foreground objects on relatively static backgrounds.

For machine learning problems, a common challenge is the selection of models and training parameters. Several research efforts have utilized visualization to compare the performance of multiple models trained with different parameters, for example, Squares [20], Model Tracker [1], and Ensemble Matrix [23]. However, a limitation is that this work depends on conventional accuracy measurements to provide evaluation data such as false positives, false negatives, and true positives. Such evaluation requires labeled data. As we have noted, a requirement for application developers is that insights are needed into detector performance even when the data is not labeled.

From a more general standpoint, our work falls within the family of spatial-temporal visualizations. See Andrienko's book for a comprehensive survey [2]. Within this research, our work most closely resembles slit-tear visualizations [24], which make use of pixels to reveal changes in a video frame over time. However, slit-tear visualizations are designed for static cameras. Our requirements are for moving cameras and also include the need to visually represent the output of the object recognizer. Recently, Buchmuller *et al.* have proposed MotionRug, which lays out moving objects in a 2D space along a 1D domain [6]. The visual compression of complex movement of objects over time into a 2D representation is a goal that we share. However, MotionRug is designed to visualize swarm behavior, where it is significant to show such features as speeding up and slowing down for the swarm as a whole. Our problem differs significantly in that we are interested in revealing the performance of recognition of individual objects over time in a moving camera scenario.

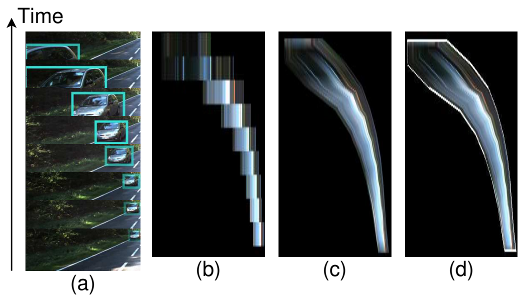

Figure 1: An illustration of how Space-Time Slicing works to visualize object recognition chains. (a): A sampling of a sequence of video frames recognizing a car (enclosed by bounding boxes with cyan color). (b): An extrustion of the middle rows of pixels in each bounding box. (c): The continuous smoothing of the bounding boxes of (b) via texture mapping. (d): Adding extra outline to enclose an object chain.

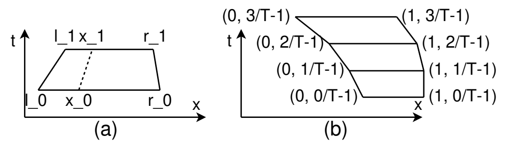

Figure 2: Geometric representation of space-time slices. (a): Connecting slices with normalized coordinates. The two vertical lines represent two slices, which has x coordinates from l_0 to r_0 and from l_1 to r_1. After linearly normalizing both $[l_0, r_0]$ and $[l_1, r_1]$ to [0, 1], the two points x_0 and x_1 have the same normalized coordinates. Thus the color along their connected line will be linearly interpolated from the colors of x_0 and x_1. (b): Space-time slices of an object chain with $T = 4$ frames. The 2D coordinates next to each vertex show its texture coordinates.

3 Space-time Slicing

Property of Driving Video Sequences We start with some observations about video sequences being captured from a stable camera on a vehicle moving on a ground plane. Movement of objects with respect to the camera frame will generally be from side-to-side, not top-to-bottom. All true objects should move smoothly with respect to the camera—they shouldn't suddenly appear in the middle of the frame or exit in the middle of the frame unless occluded by another object. They might enter at the center of the camera frame and exit at the edges (oncoming traffic, traffic being overtaken) or appear at the edges and exit at the edges (pedestrians, crossing traffic at an intersection). We should expect asymmetry since oncoming traffic will generally be on just one side or the other. If we could track and visualize the objects across frames, then these patterns should be recognizable and anomalies in these patterns could suggest problems with the recognizer or processor. Gaps in the tracks could indicate false negatives; isolated short tracks could indicate false positives; abrupt color changes along a track or branching of a track might indicate occlusion or issues with the tracking algorithm itself.

Visual design Visualizing video sequences as a 3D volume would be one approach [8]. If we were to display a video sequence as a 3D volume where x and y dimensions represent the frame space and z represents time, a true object should follow a continuous, smooth trajectory in this 3D volume. Anomalies suggested in the previous paragraph could be made apparent if they could be made visible. However, well-known problems of clutter and occlusion in 3D visualization would be very difficult to overcome. And the amount of visual data would be overwhelming. It can be tedious to find the optimal view points, and the most salient objects might be hidden within the volume.

Our approach, which we call Space-Time Slicing, is to flatten each 2D frame into as little as one horizontal line of pixels and then stack the lines vertically according to time. In this design, time will not flow left-to-right as in conventional designs. However, movement of objects will flow horizontally as is the case in the actual video sequences, and such movements are particularly significant for detecting object recognition performance.

The data that we visualize is as follows. An object detection algorithm (or a particular parameterization of an algorithm) locates and classifies foreground objects in all video frames in a video sequence. Our assumption is that the detection algorithm marks foreground objects with bounding boxes. Each bounding box is associated with a corresponding object class and a confidence score. Any object recognizer that is consistent with these assumptions is suitable for our visualization methods. After running the recognizer, we then apply a standard object tracking algorithm to link bounding boxes of potentially identical objects across video frames. Hereafter we will refer to the linked bounding boxes of an object as an object chain.

Given object chains and our original video frame sequence, our algorithm slices through the 3D spacetime volume and projects the slices to a 2D space comprised of horizontal lines revealing the positions of objects in x and their movement in time in y. Fig. 1 illustrates how our method works. Fig. 1 (a) shows a sampling of a sequence of video frames where a silver car approaches the moving camera and moves past the camera on its left-hand side. The earliest to latest video frames are stacked from bottom to top. The detections of the sliver car are enclosed with bounding boxes, which are shown in cyan color. The vehicle in oncoming traffic first appears at the bottom of the visualized trajectory and then as time passes it moves to the left-hand side of the frame and gets larger before it disappears. Its horizontal position relative to the camera viewpoint moves in time from right to left. For each bounding box, our algorithm then slices through the middle of each image patch. Fig. 1 (b) shows the extrusions of the sliced middle rows of all bounding boxes, consistent with their original horizontal locations. (Note that our visualizations in practice show each of these rectangles as a single line of pixels. We extrude them here to illustrate the relationship of the slices to the original objects.)

The visualization in Fig. 1 (b) exhibits a problem with blocking artifacts, particularly if each rectangle were reduced to a single row of pixels. To resolve this issue, we propose a novel interpolation scheme to fill the space between adjacent sliced rows of the same object. The idea is that for each point on the sliced row, which corresponds to a 3D location on a foreground object, we find the closest point in the adjacent sliced rows, and plot a line between the two points with color that is linearly interpolated from one point to the other. This interpolation scheme is based on the assumption that between adjacent video frames, both the camera parameters (location and orientation) and object (appearance, size, and location) change smoothly. Within the time interval between the two video frames, if a point on the object surface is always visible to the camera, its 2D locations on the video frame should form a smooth trajectory and the color along this trajectory should transit smoothly too.

Because finding the corresponding location of the same object at different video frames can be computationally expensive, we approximate this idea as follows. First, as object tracking already links the sliced rows of an object chain, we linearly normalize the x range of each sliced row to the range of [0, 1], and connect the pixels with the same normalized coordinates, as shown in Fig. 2 (a). Between the connected points, the colors are linearly interpolated based on the y coordinates, which are essentially the timestamps of video frames. Fig. 1 (c) shows how the rectangles of Fig. 1 (b) can be smoothly connected. In this case, the white spots of all sliced rows are connected, providing an extra cue for the underlying motion.

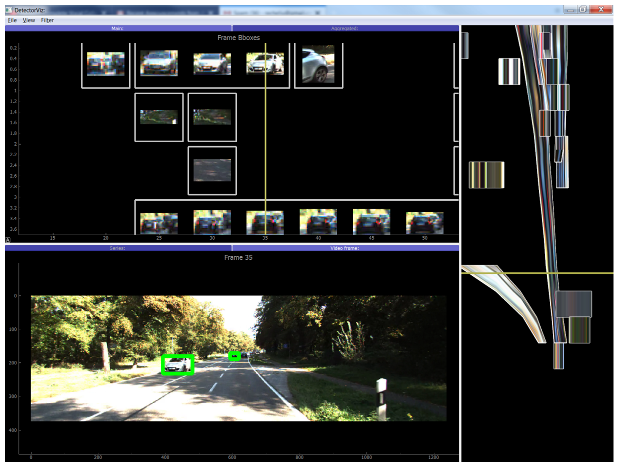
Figure 3: Zoomed-in view of Space-Time Slices (on the right) around the rectangle in Fig. 4 (c). The yellow horizontal line in the right view corresponds to the video frame shown in the bottom left view, showing that two vehicles have been detected. The top left view shows the neighboring object chains. Again the yellow line (vertical in this case) corresponds to the same frame indicated in the other views. The yellow vertical line intersects two object chains, each of which is for a detected vehicle. Other isolated image patches show false positives in nearby video frames.

Since an object video sequence can contain multiple objects, it is highly possible that multiple objects can be close to one another in the spatial domain, which can lead to cluttered space-time slices. This can also occur if the object detector incorrectly detects the same region multiple times. To remedy this issue, we can also plot extra outlines to enclose an entire object chain. An example is illustrated in Fig. 1 (d), which uses white outlines to enclose the slices in Fig. 1 (c). While motion-aware interpolation of object chains indicates how different objects move through time, object-chain outlines provide extra cues to differentiate the objects. All figures of the use case in Sect. 4 show the object chain outlines along with the interpolated colors. Users can in fact interactively show or hide either of these features.

The visual features discussed so far can help users to determine how the object recognition system is performing through visually highlighting object chains over time. Such cues can help users to see the system's differentiation of objects, even if the objects are close to one another in the video frame. However, it is still possible that objects occlude one another in fact or as a result of the recognizer, leading to overlapping space-time slices. To resolve the visibility order, our algorithm places the slices of bounding boxes with lower confidence score closer to the viewpoint. The rationale is to bring lower confidence objects to the attention of the user since they are often false positives and should be reviewed further.

Implementation With the concepts above, we can implement Space-Time Slicing with conventional computer graphics pipelines like OpenGL. Given an object chain across T video frames, our algorithm generates a 3D mesh where the x, y, and z coordinates represent the horizontal location, the time, and the score, respectively. Fig. 2 (b), for instance, shows the 3D mesh for 4 bounding boxes. For each bounding box enclosed by 2D images points (l_i, b_i) and (r_i, t_i) at frame i, the mesh has two vertices (l_i, i, s) and (r_i, i, s) where s is the score of this bounding box. Between every two consecutive frames, the mesh contains a trapezoid to connect their vertices, as shown in Fig. 2 (b). During the rendering, the camera is placed such that the z coordinate will be used for depth testing. As the z coordinate represents the score, the fragments of bounding boxes with lower confidence scores will be displayed in front of frames with higher scores. To connect the sliced rows, we apply texture mapping to map the sliced image rows to the 3D mesh. We first resize the sliced rows to a fixed width, and stack the resized slices vertically in the order of time to form the 2D texture. The texture coordinates per vertex are assigned as follows: If the i-th frame is the t-th one within this object chain, the vertical coordinate of its two vertices (l_i, i) and (r_i, i) will be $(t-1)/(T-1)$, as shown in Fig. 2.

User interaction Since Space-Time Slicing vertically compresses each frame of the video into a line that may be only one pixel high, is important to provide links from lines in the Space-Time Slicing views to other views of the data. Users may want to see the detected object(s) in the full context of video frames in which they were detected. This is especially important when multiple objects are close or even overlap on the same image row, which can lead to cluttered slices. It may be also useful to see an object or objects recognized in one frame in the context of their multi-frame object chains.

Fig. 3 shows a screenshot of our current prototype. A yellow horizontal line in the right-hand pane indicates the corresponding video frame in the lower left pane. Multiple objects are outlined and visible, even though they are small. The objects in the bottom-left frame are part of the chains represented in the top-left view above it. This pane organizes the recognized objects' image patches into small multiples of normalized size where the x-axis represents time. We align all image patches of the same object chain in the same row so the user can locate the same objects over time. Multiple rows imply different object chains. In this example, it appears that the object detector has correctly recognized two vehicles in the distance. When users click on the view of space-time slice, the top-left view will automatically scroll so the clicked time step is horizontally shifted to the middle. Such a linking scheme can help users to interactively examine different object chains, even when their slices are close.

4 Use Case

This section describes how Space-Time Slicing can be used to understand the impact of different parameters when developing and deploying an object detector. The object detector used as the example in this paper is the Single Shot Detector (SSD) [17], chosen because the source code and pre-trained parameters are publicly available. It is still considered a state-of-the-art object detection algorithm in terms of both speed and accuracy. The video sequences we use are part of the KITTI video collection [11], which were captured from car cameras under various driving scenarios (roads, university campuses, cities, suburban areas, etc.). The KITTI videos are named by the capture date and the order captured on that date. For instance, the name *KITTI-2011/09/29-0004* refers to the fourth video sequence captured on 2011/09/29.

4.1 Evaluating among differing preprocessing and training options

One preprocessing parameter for object detectors is the input image size in pixels. When training an object detector, the training images are first resized and/or cropped to a fixed resolution. This is especially crucial for object detectors that are based on CNN (Convolution Neural Networks). Since the training of CNNs essentially aims to learn 2D image filters to detect image features, the trained filters depend on the image resolution. The authors of SSD released two versions that were trained with different image resolutions, which are denoted by us as *SSD-300-PASCAL* and *SSD-512-PASCAL*. The numbers 300 and 512 in the names suggest that the detectors were trained in images of 300×300 and 512×512 pixels, respectively.

Fig. 4 (a) and (b) show the space-time slices on a video sequence *KITTI-2011/09/29-0004* with SSD-300-PASCAL and SSD-

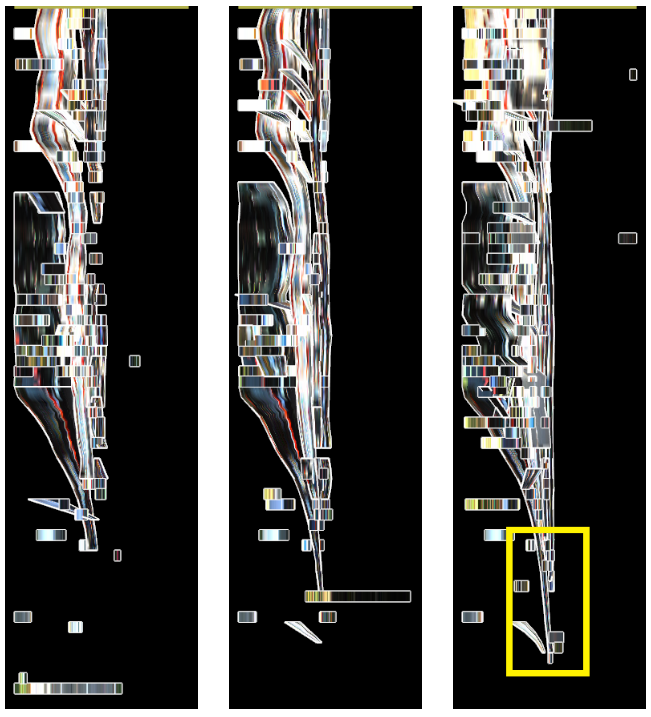

Figure 4: Space-Time Slicing of sequence KITTI-2011/09/29-0004 for detectors SSD-300-PASCAL (a), SSD-512-PASCAL (b), and SSD-300-KITTI (c). Here we sample every 5 video frames. For illustration purpose, we manually add a yellow rectangle on (c) to highlight the time steps when SSD-300-KITTI started to detect a small car.

512-PASCAL, respectively. With Space-Time Slicing, users can effectively overview and compare the performance on the extended video sequences. One use is the spotting of false positives. The space-time slices of SSD-512-PASCAL in Fig. 4 (b) shows few colored pixels on the right compared to Fig. 4 (a). We suspected, and confirmed, that this reflects ground truth since when this video sequence was captured, no objects appeared to the right of the moving vehicle. The space-time slices for detectors on 300×300 pixels exhibited more false positive, indicated by more color pixels on the right side of the visualization in Fig. 4 (a). We confirmed this by highlighting the slices in the visualizations, e.g., the lines in the lower right of Fig. 4 (a) and checking the corresponding video frame views. Space-Time Slicing can also indicate which video frames could contain false positives, such as the isolated horizontally long colored rows near the bottom of Fig. 4 (a) and (b). This can help users to check the corresponding video frames without iterating through other frames.

Fig. 4 also indicates that SSD-512-PASCAL can detect moving objects earlier than SSD-300-PASCAL. For instance, the trail at the bottom center in Fig. 4 (b) is longer than that in Fig. 4 (a), meaning that the corresponding vehicle is detected earlier (25 frames earlier in this example). This is because the vehicle or vehicles were approaching the camera and were initially small, occupying few pixels. They become larger over time. With larger image resolution, SSD-512-PASCAL can detect smaller objects than SSD-300-PASCAL. In sum, we can see that SSD-512-PASCAL appears to be performing better both in fewer false positives and higher rates of small object detection than SSD-300-PASCAL with just a quick look at these visualizations of unlabled data.

Another parameter to evaluate is the effect on detector performance of subsequent training rounds with new data. The publicly available systems *SSD-300-PASCAL* and *SSD-512-PASCAL* were trained on the publicly available dataset PASCAL VOC [9]. The scenes and object categories of PASCAL VOC are more general than those found in KITTI, which are exclusively driving-oriented. Also, the footage of PASCAL VOC does not all come from fixed car-mounted cameras. This issue is typical when adapting an existing object detector to a new target scenario. When adapting these models to new domains, their performance will almost always be sub-optimal, thus requiring *fine-tuning*. Fine-tuning means initializing a model that pre-trained parameters and resuming the training with new data for the targeted application. Fine-tuning has been shown to improve accuracy [25].

Fig. 4 (c) illustrates the result of fine-tuning *SSD-300-PASCAL* with the training images of KITTI. Our fine-tuning process followed the same procedure and configuration used to train SSD with PASCAL datasets. The KITTI training images were resized to 300×300 pixels, and thus the model with these fine-tuned parameters is denoted as *SSD-300-KITTI*. Comparing Fig. 4 (c) to Fig. 4 (a), we observe that the bounding boxes were detected earlier and that the object chain swooshes are generally more numerous and continuous. This is a likely indication that the additional training helped to identify objects at smaller sizes and also had fewer false negatives since most trails don't have the gaps that are visible in Fig. 4 (a). We can confirm the correctness of the new object trail marked with a yellow rectangle in Fig. 4 (c) by checking other linked views, which we showed earlier in Fig. 3. The portion of the Space-Time Slices shown in the rectangle are zoomed up in the right hand side of Fig. 3. On the other hand, comparing to Fig. 4 (b), Fig. 4 (c) contains more stray blobs, representing bounding boxes that cannot be tracked. After checking the detected bounding boxes, we confirm that they are actually false positives. In sum, our fine-tuning so far has apparently improved the ability to detect small true objects, even though it still can generate false positives when the image resolution is limited. Our next step might be to retrain *SSD-512-PASCAL* with our KITTI data.

4.2 Setting threshold values

Once an object detector has been trained, its deployment still requires further tuning, especially the score threshold. While raising the threshold is used to discard bounding boxes with low scores, the result of course may be to discard correctly identified objects. Typically, the selection of score thresholds requires trial-and-error, especially when ground truth is absent. Without ground truth, the R&D engineer may have to watch test video sequences in multiple iterations and it is difficult to know how to find the right balance.

With Space-Time Slicing, users can overview the impact of different thresholds over the entire video sequence. Also, because Space-Time Slicing places the slices of lower confidence bounding boxes closer to the viewer, users can know which bounding boxes in which frame are discarded without watching the video sequence frame-by-frame.

Fig. 5 shows an example to select the score threshold via Space-Time Slicing. Fig. 5 (a) shows the space-time slices of all detected objects with no threshold restrictions. The isolation of some colored slices indicates that their bounding boxes could be false positives, since they cannot be tracked. In our prototype, users can interactively adjust the score threshold via a slider-like GUI widget. Because Space-Time Slicing can be implemented via graphics pipeline, the slices with score lower than the threshold can be efficiently hidden without re-generating the slices. Instead, by simply adjusting the near distance of the camera setting, these slices can be automatically hidden by the graphics pipeline, which can be done in real time.

Fig. 5 (b) shows the space-time slices with a score threshold of 0.2, which can discard most short slices. To understand why there are still isolated space-time slices, we examine video frames and bounding boxes near frame 241, which is highlighted by the yellow

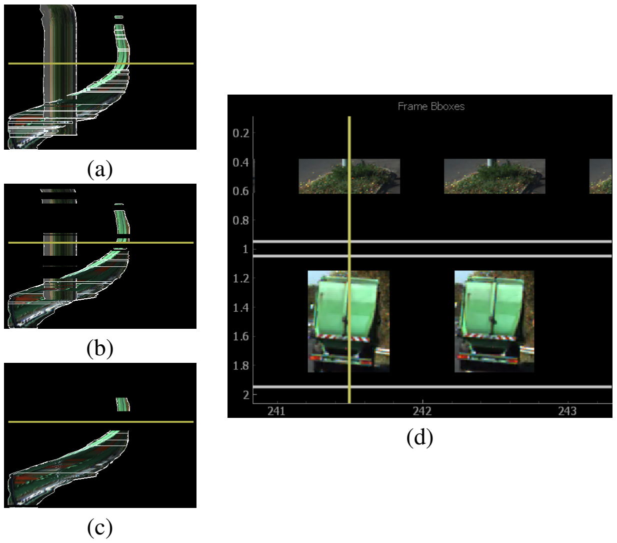

Figure 5: Adjusting of score thresholds for the detected bounding boxes in video sequence KITTI-2011/09/26-0029 by detectors SSD-512-PASCAL. (a) - (c): The space-time slices with score higher than lower bound 0.0, 0.2, and 0.3, respectively. The yellow line in (c) indicates frame 241, which is also highlighted in the view of object chains in (d). The top image patches in (d) show plants on the road, which are the straight trail in (a).

horizontal line. Fig. 5 (d) shows the object chain view in which the image patches of all object chains appear that are being tracked in frame 241. These image patches show that one of the object chains is comprised of plants on the road. These are false positives that a higher threshold would discard. When we lift the score threshold to 0.3, as shown in Fig. 5 (c), most of the retained space-time slices show smooth trajectories, suggesting that we have eliminated most false positives. However, we have inadvertently eliminated true positives as seen by the fact that there is now a gap where the yellow line is showing that there should be a truck being tracked. The development process must continue.

5 Conclusion

In this paper, we have presented Space-time Slicing, a visualization method useful for understanding the performance of object detectors and the impact of various parameters on video sequences. Space-Time Slicing visualizations can provide an overview of recognized objects and their significant movement in video sequences. They can be used in conjunction with linked views of object chains and individual video frames to confirm the performance of object detection in key parts of the video and draw conclusions about the comparative performance of the object detectors under different pre-processing options, training regimes, and threshold settings.

A limitation of our current implementation is that each frame of video requires one horizontal pixel line, so the vertical height of any particular visualization in pixels will be tied to the total number of frames in the video. It is not suitable for summarizing more than a few minutes of video at standard frame rates in one screen. Future work could allow the interactive adjustment of video frames to show different levels of detail, based on the view scale and screen resolution. We will also like to integrate space-time slicing with other time-varying visualization approaches to reveal extra information, such as co-occurence of objects and events [16].

References

[1] S. Amershi, M. Chickering, S. Drucker, B. Lee, P. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *CHI*, 2015.

[2] N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer-Verlag, Berlin, Heidelberg, 2005.

[3] E. P. Bennett and L. McMillan. Computational time-lapse video. *ACM Transactions on Graphics*, 26(3), 2007.

[4] R. Borgo, M. Chen, B. Daubney, E. Grundy, G. Heidemann, B. Höferlin, M. Höferlin, H. Leitte, D. Weiskopf, and X. Xie. State of the art report on video-based graphics and video visualization. *Computer Graphics Forum*, 31(8):2450–2477, 2012.

[5] R. P. Botchen, S. Bachthaler, F. Schick, M. Chen, G. Mori, D. Weiskopf, and T. Ertl. Action-based multifield video visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):885–899, 2008.

[6] J. Buchmüller, D. Jäckle, E. Cakmak, U. Brandes, and D. A. Keim. MotionRugs: Visualizing collective trends in space and time. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):76–86, 2019.

[7] C. D. Correa and K.-L. Ma. Dynamic video narratives. In *ACM Transactions on Graphics*, pp. 88:1–88:9, 2010.

[8] G. Daniel and M. Chen. Video visualization. In *Vis '03: Proceedings of the IEEE International Conference on Visualization*, pp. 54–, 2003.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[10] S. Fels, E. Lee, and K. Mase. Techniques for interactive video cubism. In *MM '00: ACM International Conference on Multimedia (Poster Session)*, pp. 368–370, 2000.

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[12] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[13] D. B. Goldman, B. Curless, D. Salesin, and S. M. Seitz. Schematic storyboarding for video visualization and editing. *ACM Transactions on Graphics*, 25(3):862–871, 2006.

[14] T. Lee, A. Chaudhuri, F. Porikli, and H. Shen. Cyclestack: Inferring periodic behavior via temporal sequence visualization in ultrasound video. In *PacificVis '10: Proceedings of the IEEE Pacific Visualization Symposium*, pp. 89–96, 2010.

[15] P. A. Legg, D. H. S. Chung, M. L. Parry, M. W. Jones, R. Long, I. W. Griffiths, and M. Chen. Matchpad: Interactive glyph-based visualization for real-time sports performance analysis. *Computer Graphics Forum*, 31(3pt4):1255–1264, 2012.

[16] J. Li, S. Chen, K. Zhang, G. Andrienko, and N. Andrienko. COPE: Interactive exploration of co-occurrence patterns in spatial time series. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[18] M. L. Parry, P. A. Legg, D. H. S. Chung, I. W. Griffiths, and M. Chen. Hierarchical event selection for video storyboards with a case study on snooker video visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1747–1756, 2011.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[20] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, Jan 2017.

[21] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[22] K. Sunkavalli, W. Matusik, H. Pfister, and S. Rusinkiewicz. Factored time-lapse video. *ACM Transactions on Graphics*, 26(3), 2007.

[23] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan. Ensemblematrix: Interactive visualization to support machine learning with multiple classifiers. In *CHI*, CHI '09, pp. 1283–1292, 2009.

[24] A. Tang, S. Greenberg, and S. Fels. Exploring video streams using slit-tear visualizations. In *AVI '08: Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 191–198, 2008.

[25] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, pp. 3320–3328, 2014.