Specifying and Evaluating Quality Metrics for Vision-based Perception Systems

Anand Balakrishnan*, Aniruddh G. Puranic*, Xin Qin*, Adel Dokhanchi†,
Jyotirmoy V. Deshmukh*, Heni Ben Amor†, Georgios Fainekos†
* University of Southern California. Email: {anandbal, puranic, xinqin, jdeshmuk}@usc.edu,
† Arizona State University. Email: {adkohanc, hbenamor, fainekos}@asu.edu

***Abstract*—Robust perception algorithms are a vital ingredient for autonomous systems such as self-driving vehicles. Checking the correctness of perception algorithms such as those based on deep convolutional neural networks (CNN) is a formidable challenge problem. In this paper, we suggest the use of Timed Quality Temporal Logic (TQTL) as a formal language to express desirable spatio-temporal properties of a perception algorithm processing a video. While perception algorithms are traditionally tested by comparing their performance to ground truth labels, we show how TQTL can be a useful tool to determine quality of perception, and offers an alternative metric that can give useful information, even in the absence of ground truth labels. We demonstrate TQTL monitoring on two popular CNNs: YOLO and SqueezeDet, and give a comparative study of the results obtained for each architecture.**

***Index Terms*—Temporal Logic, Monitoring, Autonomous vehicles, Perception, Image processing, Quality Metrics**

I. Introduction

Recent developments in GPU-based computing and fundamental advances in stochastic optimization have led to an explosion in the applications of Deep Learning (DL) techniques using Convolutional Neural Network (CNN) architectures to real-time image recognition problems [1]. Such techniques are being used in *safety-critical* applications such as self-driving vehicles [2]. Testing of these systems is largely based on either (a) measuring the recognition error on a pre-recorded data-set, or (b) running actual driving tests on the road with a backup human driver and focusing on the disengagements, i.e. events where the autonomous vehicle returns control back to the driver [3], [4]. Even in the latter case, 37.9% of disengagements in the 2016 disengagements report filed with the California DMV were attributed to perception discrepancy (i.e. a situation in which the self-driving car's sensors are not correctly perceiving an object). Finally, consider the recent Uber accident [5], [6]. The analysis of this accident as stated in the NTSB report [6] states, "as the vehicle and pedestrian paths converged, the self-driving system software classified the pedestrian as an unknown object, as a vehicle an then as a bicycle with varying expectations of future travel paths." Clearly, even if different sensors had detected the pedestrian, the perception system was not robust enough to assign a single consistent object class to the pedestrian, and, hence, it was not able to predict her future path. We argue that the Uber accident was not just due to insufficient testing or poor engineering choices, but rather due to the lack of ways to formally express what should be the performance and guarantees provided by a perception system. It is clear that without formal requirements, any behavior can be deemed acceptable. In this paper, we argue for an urgent need for techniques to formally state specifications on the *quality* of a perception algorithm.

Informally, the *quality* of a perception algorithm is a judgement on how well the perception algorithm respects certain spatio-temporal logical relations within objects in a scene. We provide a formal framework that facilitates not only specification of such relations, but also enables efficient algorithms to check the specifications against the output of the given perception algorithm. Such a formal framework would enable creating a precise and real-life set of requirements that need to be met by any learning-based model. Furthermore, these specifications can be used as real-time monitors in an actual vehicle to serve as indicators of possible failures in perception.

Our spatio-temporal quality requirements are expressed in a logic that we call Timed Quality Temporal Logic (TQTL). This is an extension of Timed Propositional Temporal Logic (TPTL) [7], [8]. TPTL is a logic that uses explicit clock variables to reason about logical relations between Boolean-valued propositions across time. In our setting, the role of "time" is played by the frame number of a given video, and TQTL allows us to compare relations between object labels across different frames.

Rest of the paper is organized as follows. In Section II, we review the syntax and semantics of TQTL. The main contribution of this paper is in section III, where, we demonstrate the usefulness of TQTL by running it on two case studies. The first case study uses the CNN Yolo to perform object recognition on driving data obtained from the Kitti vision benchmark suite. The second case study uses the CNN SqueezeDet on the same data. Finally, we compare the performance of the two networks with respect to the quality metric. This gives us yet another use of the quality metric: it serves as a standard for benchmarking the performance of perception algorithms.

II. Timed Quality Temporal Logic

We assume that perception algorithms considered in this paper are based on real-time object recognition. We focus on convolutional neural networks (CNN) that take as input a video signal, i.e. a sequence of frames ordered by frame number, and

978-3-9819263-2-3/DATE19/©2019 EDAA

output a stream of structure data objects. We call such a stream the *data stream.*

Definition II.1 (Data Stream, Frames and Data Objects)**.** A *data stream* $\mathcal{D}$ is a sequence of frames $\langle \mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_n \rangle$. The index t of *frame* $\mathcal{D}_t$ is called the *frame number*, assumed to be a non-negative integer. A frame $\mathcal{D}_t$ is a set of m_t *data objects* $\{d_1, \ldots, d_{m_t}\}$, where each d_i is an element of some set $\mathcal{O}_i$, also known as the data domain.

Example II.1. Consider the following data stream:

$$\begin{array}{rl} \mathcal{D}_0: & d_1 : ((\texttt{ID}, 1),\ (\texttt{class}, car),\ (\texttt{pr}, 0.9),\ (\texttt{bb}, B_1)) \\ \mathcal{D}_1: & d_1 : ((\texttt{ID}, 1),\ (\texttt{class}, car),\ (\texttt{pr}, 0.9),\ (\texttt{bb}, B_1)) \\ & d_2 : ((\texttt{ID}, 2),\ (\texttt{class}, car),\ (\texttt{pr}, 0.8),\ (\texttt{bb}, B_2)) \\ \mathcal{D}_2: & d_1 : ((\texttt{ID}, 1),\ (\texttt{class}, car),\ (\texttt{pr}, 0.9),\ (\texttt{bb}, B_1)) \\ & d_2 : ((\texttt{ID}, 3),\ (\texttt{class}, pedestrian),\ (\texttt{pr}, 0.6), \\ & \quad (\texttt{bb}, B_3)) \ldots \end{array}$$

Consider the frame $\mathcal{D}_2$: (d_1, d_2), where each d_i is defined as a tuple of key-value pairs. For example, d_1 is: $(\ (\texttt{ID}, 1), (\texttt{class}, car), (\texttt{pr}, 0.9), (\texttt{bb}, B_1)\ )$, where the value for the key ID contains a unique (within the frame) positive integer indicating the object id, key class is a string specifying the object type, the key pr points to a real number in $[0, 1]$ indicating the probability of car indeed being the label of this d_1, and $B_1 \in \mathbb{Z}^4$ is a bounding box (bb) indicating the top, left, bottom, and right pixels indicating corners of an axis-oriented rectangle that d_1 represents.

Definition II.2 (Information Retrieval Function and Set of Objects Function)**.** An information retrieval function $\mathcal{R}$ maps a frame number and an positive integer representing object id to a given data object d. If $d = ((key_1, val_1), \ldots, (key_\ell, val_\ell))$, we use the notation $d.key_k$ to denote the the value val_k. The function $\mathcal{S}$ maps a given frame number to the set of object identifiers (IDs) in that frame.

Example II.2. Consider Example II.1, $\mathcal{R}(\mathcal{D}_2, 3).\texttt{class}$ refers to the class-label of the data object with $\texttt{ID} = 3$ in the frame $\mathcal{D}_2$ (which is $pedestrian$). Also, $\mathcal{S}(\mathcal{D}_2) = \{1, 3\}$.

Remark II.1 (Tracking Assumption)**.** We make a crucial assumption that any unique object detected by the perception video gets assigned a unique ID. Furthermore, we assume that the same ID is used across different frames. In other words, we assume that the perception algorithm is capable of tracking objects.

A. Timed Quality Temporal Logic

To define quality of perception tasks, we need the ability to compare data objects in different frames of a video. Similarly, we need to have the ability to reason about a dynamic number of data objects in each frame. These tasks cannot be achieved by logics such as STL (Signal Temporal Logic) [9], [10] that are used for specifications of cyber-physical systems.

Hence, in [11], we introduced Timed Quality Temporal Logic (TQTL). TQTL is an extension of Timed Propositional Temporal Logic (TPTL) [7] to reason about arbitrary relations between data objects. TPTL is a logic that extends propositional logic with the notion of clock variables, and is able to specify interesting properties over timed (Boolean-valued) traces. It is more expressive than STL as it allows comparing values of signals at different time instants. However, TPTL only supports traces over a pre-defined vocabulary of atomic propositions. In perception tasks, we can have a variable number of data objects in each frame. Thus, we extend TPTL to TQTL to provide additional support for quantification over objects in a given frame.

Definition II.3 (Scoring Function, Quality Predicates)**.** We assume that we have a finite set of predicates $\mathcal{P} = \{\mu_1, \ldots, \}$, where each μ_j is a Boolean-valued predicate over attributes of a data object. In general, μ_j has the following form:

$$\mu_j \equiv f_{\mu_j}(t_1, \ldots, t_{n_1}, id_1, \ldots, id_{n_2}) \sim c \qquad (1)$$

Here, $t_1, \ldots, t_{n_1}$ are *frame number variables*, and $id_1, \ldots, id_{n_2}$ are integers representing object IDs. The function f is known as a *scoring function* that given an evaluation for $t_1, \ldots, t_{n_1}$ and $id_1, \ldots, id_{n_2}$, computes a real valued score (using some subset of object attributes that it defines). The operator $\sim$ is a comparison operator, i.e. $\sim \in \{<, \leq, >, \geq, =, \neq\}$, and c is a real number.

Example II.3. Consider the function f defined as $(t, u, id) \mapsto \mathcal{R}(\mathcal{D}_t, id).\texttt{pr} \times \mathcal{R}(\mathcal{D}_u, id).\texttt{pr}$. Given a valuation for variables t, u, and the variable id, this function computes the product of the probabilities of the objects with the ID as specified by the valuation of id. Then, $\mu \equiv f(t, u, id) > 0.9$ is a predicate that checks whether the value of the scoring function is greater than 0.9.

We are now ready to present the syntax of TQTL:

Definition II.4 (TQTL Syntax)**.** A TQTL formula φ over a finite set of predicates $\mathcal{P}$, a finite set of frame number variables ($\mathcal{V}_t$), and a finite set of object ID variables ($\mathcal{V}_{id}$) is inductively defined according to the following grammar:

$$\begin{array}{rl} \varphi ::= & \top \mid \mu \mid t.\varphi \mid \exists id@t, \varphi \mid \forall id@t, \varphi \mid \\ & t \leq u + n \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U} \varphi_2 \end{array} \qquad (2)$$

In the above definition, $\mu \in \mathcal{P}$, $\top$ is the predicate *true*, $t, u \in \mathcal{V}_t$, $n \in \mathbb{Z}^{\geq 0}$, $id \in \mathcal{V}_{id}$, and $\mathbf{U}$ is the until operator in temporal logics such as STL. The time constraints of TQTL are represented in the form of $t \leq u + n$. The *freeze frame quantifier* $t.\varphi$ assigns the current time to time variable t before processing the subformula φ. The quantifiers $\exists$ and $\forall$ respectively existentially and universally quantify over the object IDs in a given frame. The semantics of TQTL is defined using an valuation function ν that maps every variable in $\mathcal{V}_t \cup \mathcal{V}_{id}$ to some number in $\mathbb{N}$. The valuation function provides an environment in which to interpret the given TQTL formula.

Definition II.5 (TQTL Semantics)**.** The *quantitative semantics* of a TQTL formula are defined in terms of a function $[\![\varphi]\!]$ that maps a data stream $\mathcal{D}$, a frame number τ, and a valuation function ν (that assigns values to the frame number and ID variables) to a real number. The convention that we use in the semantics below is that Roman letters t, u denote frame number variables, while the Greek symbols τ, τ', τ'' denote actual time values. The function $\llbracket \cdot \rrbracket$ is recursively defined as follows:

φ	$\llbracket \varphi \rrbracket (\mathcal{D}, \tau, \nu)$
$\top$	$+\infty$
μ	$\theta(\nu(f_\mu(t_1, \ldots, t_{n_1}, id_1, \ldots, id_{n_2})), c)$
$t.\varphi$	$\llbracket \varphi \rrbracket (\mathcal{D}, \tau, \nu[t \leftarrow \tau])$
$\exists id@t, \varphi$	$\max\limits_{k \in \mathcal{S}(\mathcal{D}_{\nu(t)})} \llbracket \varphi \rrbracket (\mathcal{D}, \tau, \nu[id \leftarrow k])$
$t \leq u + n$	$\begin{cases} +\infty & \text{if } \nu(t) \leq \nu(u) + n, \\ -\infty & \text{otherwise.} \end{cases}$
$\neg \varphi$	$- \llbracket \varphi \rrbracket (\mathcal{D}, \tau, \nu)$
$\varphi_1 \wedge \varphi_2$	$\min(\llbracket \varphi_1 \rrbracket (\mathcal{D}, \tau, \nu), \llbracket \varphi_2 \rrbracket (\mathcal{D}, \tau, \nu))$
$\varphi_1 \mathbf{U} \varphi_2$	$\max\limits_{\tau' \geq \tau} \min \left(\begin{array}{c} \llbracket \varphi_2 \rrbracket (\mathcal{D}, \tau', \nu), \\ \min\limits_{\tau'' \in [\tau, \tau')} \llbracket \varphi_1 \rrbracket (\mathcal{D}, \tau'', \nu) \end{array} \right)$

Here, the most important aspect is definition of the *quality of a predicate*, which is defined in terms of a *comparison* function θ and the valuation function ν applied to the scoring function f_μ. The latter, i.e., $\nu(f_\mu(t_1, \ldots, t_{n_1}), id_1, \ldots, id_{n_2})$, is abuse of notation, and is equivalent to $f_\mu(\nu(t_1), \ldots, \nu(t_{n_1}), \nu(id_1), \ldots, \nu(id_{n_2}))$. The value returned by the θ function depends on the type of the output of the f_μ function, and the comparison operator $\sim$. If f_μ returns a value in a metric space, i.e., in a space equipped with an appropriate metric m_f, and if we rewrite μ in the form $f_\mu(\cdot) > c$ or $f_\mu(\cdot) \geq c$, then, $\theta(f_\mu(\cdot), c) = f_\mu(\cdot) - c$. If f_μ returns a value over a discrete space (such as an enumeration), then, $\theta(f_\mu(\cdot), c) = +\infty$ if $f_\mu(\cdot) = c$, and $-\infty$ otherwise. We note that the definition of the comparison function over metric spaces is very similar to the one introduced for the logic Metric Temporal Logic in [12].

Further, $\nu[w \leftarrow a]$ assigns the value a into the variable $w \in \mathcal{V}_{id} \cup \mathcal{V}_t$, while leaving the values of all other variables unchanged. Finally, we say that $\mathcal{D}$ *satisfies* a TQTL specification φ if $\llbracket \varphi \rrbracket (\mathcal{D}, 0, \nu_0) > 0$. Here, ν_0 is an initial valuation to all variables. The semantics of TQTL will become clearer in the next section when we look at specific examples.

III. Experimental Results

Object detection is a trickier problem than image classification as it possibly involves detecting multiple objects in an image [13]. There are two main approaches in generic object detection, (1) region proposal based approaches and (2) regression based approaches. In the former approach, a deep neural network makes so-called 'region proposals' (portions of the image determined by selective search) and classifies each region. These approaches generally have multi-step training requirements, and are slow due to the fact that they need to perform multiple operations on the entire image multiple times. Examples of the convolutional neural network architectures that use these approaches include R-CNN [14], Fast R-CNN [15], and Faster R-CNN [16]. These approaches are also characterized by some degree of reliance on algorithms based on geometric methods in computer vision.

In contrast to the region proposal based approaches, regression-based approaches like YOLO [17] and SqueezeDet [18] formulate generic multi-object detection as regression mapping directly from the image to the bounding box coordinates and class probabilities. Moreover, these methods are single-shot detectors, that is, they only process the input image once and detect multiple bounding boxes. These methods use fixed-grid regression, a method that combines the ideas of region proposal and regression.

YOLO divides the the input image into an $S \times S$ grid and the convolution layers detect features in each individual grid cell, come up with a bounding box for the object in each cell, and combine the bounding boxes of the individual cells into a larger bounding box. The feature extraction layers are trained separately, on datasets like ImageNet [19], and the object detection layers are trained in combination with the feature detection layers. In the latest version of YOLO, a custom, fully-convolutional feature extraction layer (Darknet-53) is trained on ImageNet and the object detection layer is also a 53 layer CNN [17].

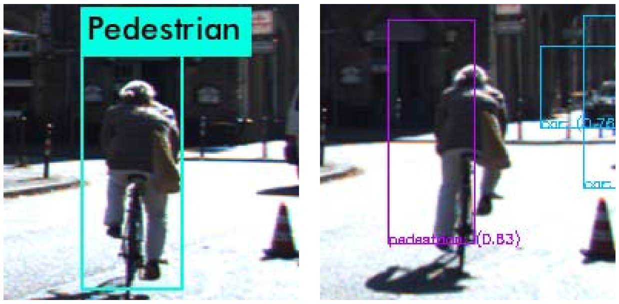
SqueezeDet [18] is similar to YOLO, as it is also a fixed-grid regression based approach where the deep neural network is trained in two stages, a feature detector and then the object detector. Where SqueezeDet differs from YOLO is in the architecture of the object detection layers. The authors of SqueezeDet define a fully-convolutional network architecture called ConvDet, that is capable of mapping from each feature detected in the grids to a bounding box with class probabilities, from which the top N predictions are outputted.

A. Experiment Overview

The two architectures used for our experiments, SqueezeDet [18] and YOLOv3 [17], were both trained on the KITTI object detection dataset [20] and evaluated on individual sessions in the KITTI raw dataset [21]. SqueezeDet was evaluated using pre-trained weights available at the code repository maintained by the authors of the original SqueezeDet paper [18] . YOLO was trained on the KITTI object detection dataset with 9 classes (Car, Van, Truck, Tram, Pedestrian, Person_Sitting or Sitter, Cyclist, Misc., and DontCare) for 10000 iterations in 12 hours.

The data streams generated by the two algorithms were monitored on a subset of the KITTI raw dataset against the TQTL specifications of the form shown in Eq. (3). For notational convenience, we introduce the notation $C(x, id)$ as an abbreviation for $\mathcal{R}(\mathcal{D}_x, id)$`.class`, and $P(x, id)$ as an abbreviation for $\mathcal{R}(\mathcal{D}_x, id)$`.pr`.

$$\begin{aligned} \varphi = & \Box(x.\forall id_1@x, (C(x, id_1) = \mathit{Cyclist} \wedge P(x, id_1) > 0.7) \\ & \rightarrow \phi) \end{aligned} \tag{3}$$



(a) YOLO Detection (b) SqueezeDet Detection

Fig. 1. An example of both the object detection algorithms failing to classify a cyclist correctly in the KITTI raw dataset session 0005, near frame 80.

The above equation monitors the property that *"at every time step, for all objects (id_1) in the frame, if the object class is $Cyclist$ with probability > 0.7, then some property ϕ holds."* One of the quality metrics that is applicable to object tracking is to check if the algorithm continues to track an object and loses confidence in the object label only at a reasonable rate. A TQTL property that can monitor this is the following:

$$\begin{aligned}\varphi_1 =&\Box(x.\forall id_1@x, (C(x, id_1) = Cyclist \wedge P(x, id_1) > 0.7)\\ &\rightarrow \Box(y.((x \leq y \wedge y \leq x + 5)\\ &\quad\rightarrow C(y, id_1) = Cyclist \wedge P(y, id_1) > 0.6))\end{aligned} \tag{4}$$

This property checks that if the object detection algorithms detect cyclists in any frame x with a probability of larger than 0.7, then for the next 5 frames, the probability that the object is a cyclist doesn't fall below 0.6. But sometimes, it is observed that cyclists are misclassified as pedestrians by both object detection algorithms, as shown in Fig. 1. This can possibly be attributed to the bicycle not being prominent in the images where the camera direction is almost parallel to the direction of the cyclist, causing the cyclist to look like a pedestrian. This can cause the specification in 4 to be violated. Hence, to monitor if the algorithms are not able to detect any cyclist, when in fact there is at least one, we also define the following TQTL specification.

$$\begin{aligned}\varphi_2 =&\Box(x.\forall id_1@x, (C(x, id_1) = Cyclist \wedge P(x, id_1) > 0.7)\\ &\rightarrow \Box(y.((x \leq y \wedge y \leq x + 5)\\ &\quad\rightarrow C(y, id_1) = Cyclist \wedge P(y, id_1) > 0.6\\ &\qquad\vee \exists id_2@y, (C(y, id_2) = Pedestrian\\ &\qquad\quad\wedge\, dist(x, y, id_1, id_2) < 40\\ &\qquad\quad\wedge\, P(y, id_2) > 0.6)))\end{aligned} \tag{5}$$

Here, the $dist$ function extracts the bounding boxes of object id_1 at frame x and id_2 at frame y for computing the center-to-center distance between the boxes. It essentially says that either the object detected is a cyclist with probability greater than 0.6 or there is some other object that is a pedestrian with probability greater than 0.6, that is less than 40 pixels distance from what was thought as a cyclist in a previous frame. This property illustrates the power of TQTL in able to compare object attributes in different frames.

TABLE I
ROBUSTNESS RESULTS OBTAINED FROM MONITORINGYOLOV3 AND SQUEEZEDET AGAINST φ_1 AND φ_2

Session ID	**SqueezeDet**		**YOLO**	
	φ_1	φ_2	φ_1	φ_2
0001	-0.14	-0.14	-0.28	-0.28
0002	0.1	0.1	0.7	0.7
0005	-0.24	-0.24	-0.29	-0.28
0009	0.7	0.7	0.7	0.7
0011	-0.13	-0.13	-0.19	-0.19
0014	-0.22	-0.22	-0.29	-0.29
0015	0.10	0.1	-0.19	-0.19
0017	-0.06	-0.06	0.7	0.7
0018	-0.02	-0.02	0.7	0.7
0019	-0.18	-0.18	-0.28	-0.28
0020	-0.17	-0.17	-0.22	-0.22
0022	0.7	0.7	0.7	0.7
0023	0.7	0.7	0.7	0.7
0027	0.7	0.7	0.7	0.7
0028	0.23	0.23	0.7	0.7

B. TQTL Monitoring Results

On monitoring the two CNN-based object detection algorithms on a subset of the KITTI raw dataset, we observed the robustness results shown in TABLE I. It can be seen here that even though our monitor takes into account misclassifying cyclists as pedestrians, there isn't significant difference between the robustness values of the algorithms. This can be attributed to the algorithms either detecting *"phantom"* objects, that is, detect objects when there are none; or the algorithms regularly fail to detect prominent objects.

To further analyze the robustness results, sessions 0005 and 0017 were chosen, as they are examples where either both algorithms consistently fail, or one performs much better than the other.

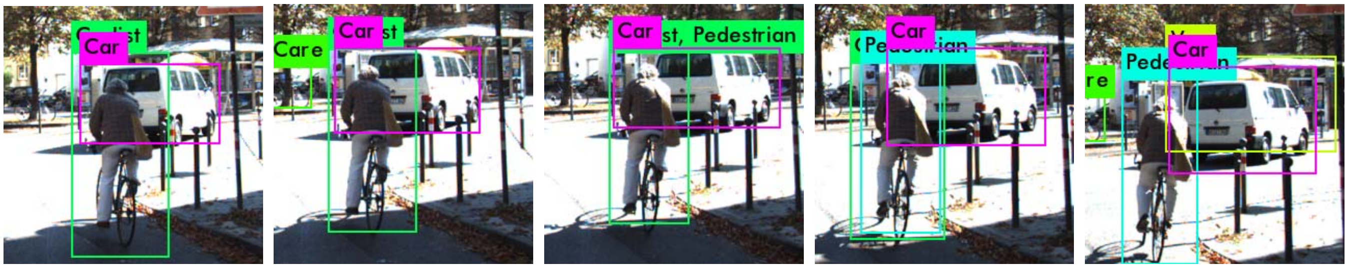
C. Analysis for YOLO

From Fig. 2 we can see that φ_1 (specified in Eq. (4)) is violated as the cyclist that YOLO is tracking is falsely identified as a pedestrian with relatively high confidence. This stream was generated from Session 0005 of the KITTI raw dataset, which shows the negative robustness in TABLE I.

Similarly, in Fig. 3, we can see that the cause for negative robustness when monitored against Formula 5 is that the probability of the cyclist being labeled correctly as a cyclist or a pedestrian is dropping below 60%, thereby violating the condition laid out by φ_2.

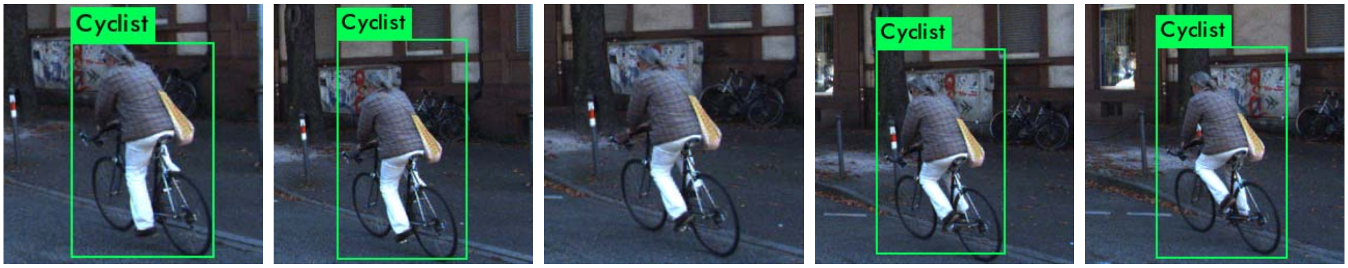
D. Analysis for SqueezeDet

The stream in Fig. 4 shows how SqueezeDet violates φ_1 (specified in Eq. (4)), as the cyclist is falsely labelled as a pedestrian with high confidence. This shows that, similar to YOLO, images where the cyclist is traveling almost parallel to the direction of the camera, the algorithm misclassifies



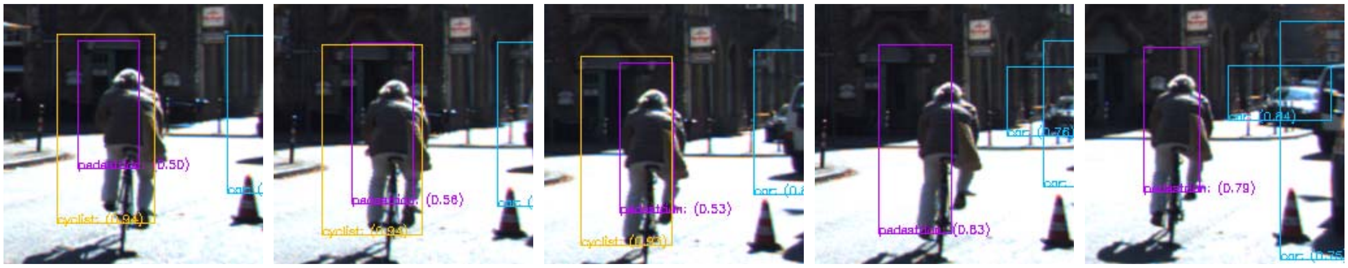
(a) 51 (b) 52 (c) 53 (d) 54 (e) 55

Fig. 2. In session 0005, YOLO misclassifies the Cyclist as a Pedestrian with high confidence (more than 75%).



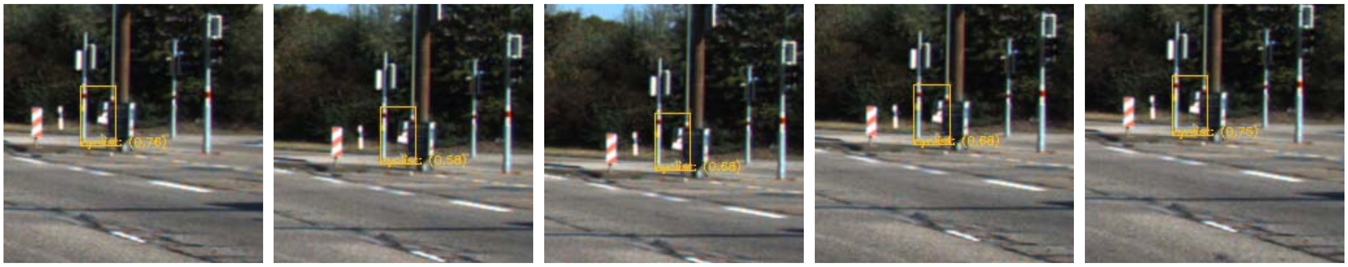
(a) 22 (b) 23 (c) 24 (d) 25 (e) 26

Fig. 3. In session 0005, YOLO intermittently detects the Cyclist with varying levels of probability (0 – 75%).



(a) 81 (b) 82 (c) 83 (d) 84 (e) 85

Fig. 4. In session 0005, SqueezeDet misclassifies the Cyclist as a Pedestrian with high confidence (more than 75%).



(a) 65 (b) 66 (c) 67 (d) 68 (e) 69

Fig. 5. In session 0017, SqueezeDet intermittently detects a non-existent Cyclist with varying levels of probability (55 – 75%).

the cyclist as a pedestrian. And while we also employed φ_2 (Eq. (5)) to monitor this misclassification, the algorithm still violates the property. This can be attributed to *"phantom"* objects being detected by SqueezeDet with high probability and suddenly not detecting it. An example of this can be seen in Fig. 5.

E. Comparative Analysis

We compared the results of YOLO and SqueezeDet on the same datasets with the same settings (such as window frame range for analysis). Our observations are:

1) Both of them mis-classify cyclists as pedestrians in several instances. This mainly seems to happen when the cyclist plane is orthogonal to the image plane. This causes the cyclist to look like a pedestrian. When the image is of the bicyclist's back, it very thin and hence makes the cyclist look like a pedestrian. This could indicate that the KITTI dataset doesn't have enough instances where the the bicyclist is captured from directly behind him or in front of him.
2) Both algorithms detect objects intermittently, that is, lose confidence in their prediction very quickly.
3) An observation made in the case of SqueezeDet is that it detects several *"phantom"* objects with high confidence, and quickly loses confidence in these false predictions.

We use the above observations to further argue the power of using TQTL-based monitoring. Testing object detection algorithms against ground truth labels would have captured some of the above observations, but would have not made the rich temporal debugging information that we were able to obtain – *without any ground truth labels*! Essentially using TQTL we were quickly able to *localize* examples of low-quality perception, and help the designer understand the ramifications of mis-classifications across frames.

IV. Conclusion and Future Work

In this paper, we reviewed Timed Quality Temporal Logic (TQTL) as a formalism to specify spatio-temporal properties of perception algorithms. We applied TQTL-based monitoring on the outputs of two popular convolutional neural network architectures used for real-time object detection (YOLOv3 and SqueezeDet). Using TQTL, we were able to find interesting examples that localize low quality results from the perception algorithm to a sequence of frames. Such information can be invaluable when debugging a perception algorithm, especially when it is used in a safety-critical context.

In future work, we wish to extend TQTL to consider spatio-temporal relations between objects. We are also interested in comparing perception algorithms that natively perform object tracking against those that do not track objects using the result of TQTL monitors. Finally, we are interested in automatically generating feedback to perception system designers from the results of quality monitoring using TQTL.

Acknowledgements This work was partially supported by the NSF I/UCRC Center for Embedded Systems and by NSF grants 1350420, 1361926, 1446730, and CCF 1837131.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] "IEEE Connected Vehicles: Google reports self-driving car disengagements," available at: http://sites.ieee.org/connected-vehicles/2015/12/15/google-reports-self-driving-car-disengagements.

[4] "Autonomous vehicle disengagement reports 2016," available at: https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2016.

[5] T. B. Lee. (2018) Report: Software bug led to death in ubers self-driving crash. [Online]. Available: https://arstechnica.com/tech-policy/2018/05/report-software-bug-led-to-death-in-ubers-self-driving-crash/

[6] National Transportation Safety Board, "Ntsb preliminary report," Tech. Rep., 2018. [Online]. Available: https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf

[7] R. Alur and T. A. Henzinger, "A really temporal logic," *J. ACM*, vol. 41, no. 1, pp. 181–204, 1994.

[8] A. Dokhanchi, B. Hoxha, C. E. Tuncali, and G. Fainekos, "An efficient algorithm for monitoring practical TPTL specifications," in *The ACM/IEEE International Conference on Formal Methods and Models for System Design, (MEMOCODE)*, 2016, pp. 184–193.

[9] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proceedings of FORMATS-FTRTFT*, ser. LNCS, vol. 3253, 2004, pp. 152–166.

[10] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications," in *Lectures on Runtime Verification - Introductory and Advanced Topics*, ser. LNCS. Springer, 2018, vol. 10457, pp. 128–168.

[11] A. Dokhanchi, H. B. Amor, J. V. Deshmukh, and G. Fainekos, "Evaluating perception systems for autonomous vehicles using quality temporal logic," in *Proc. of Runtime Verification*, 2018, pp. 409–416.

[12] G. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.

[13] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *arXiv preprint arXiv:1807.05511*, 2018.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[15] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1137–1149, 2017.

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[18] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving." in *CVPR Workshops*, 2017, pp. 446–454.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255.

[20] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3354–3361.

[21] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.