A reasoning based model for anomaly detection in the Smart City domain

Patrick Hammer[1], Tony Lofthouse[2], Enzo Fenoglio[3], and Hugo Latapie[4]

[1] Department of Computer & Information Sciences
College of Science and Technology
Temple University
Philadelphia PA 19122, USA
patrick.hammer@temple.edu
[2] Reasoning Systems Ltd.
tony_lofthouse@btinternet.com
[3] Cisco Systems Inc.
efenogli@cisco.com
[4] Cisco Systems Inc.
hlatapie@cisco.com

Abstract. Using a proprietary visual scene object tracker and the OpenNARS reasoning system we demonstrate how to predict and detect various anomaly classes. The approach combines an object tracker with a base ontology and the OpenNARS reasoning system to learn to classify scene regions based on accumulating evidence from typical entity class (tracked object) behaviours. The system can autonomously satisfy goals related to anomaly detection and respond to user Q&A in real time. The system learns directly from experience with no initial training required (one-shot). The solution is a fusion of sub-symbolic (object tracker) and symbolic (ontology and reasoning).

1 Introduction and similar work

Supervised Deep Learning has generated a lot of success in image classification, object detection and tracking. Combining this technology with a reasoning-learning system opens up new possibilities for anomaly prediction and detection. Our approach uses a Non-Axiomatic Reasoning System (NARS) which consumes tracklets provided by a Multi-Class Multi-Object Tracking (MC-MOT) system in real-time. The system infers spatial and temporal relations between events, describing their relative position and predicted path.

The multi-class and multi-object tracker was developed by Cisco Systems Inc, and the reasoning-learning component by OpenNARS [1]. We tested our approach on a publicly available dataset, 'Street Scene', which is a typical scene from a Smart City domain. The dataset is video data obtained by a street cam mounted on a building. We show the system learning to classify scene regions, such as street or sidewalk, from the typical 'behaviour' of the tracked objects belonging to three classes, such as car, pedestrian, and bike. The system is shown to autonomously satisfy a range of goals to detect and inform the user of certain anomaly types. An anomaly ontology supports various anomaly classes; location based, relational based, velocity based, or vector based. From this anomaly ontology a range of specific anomalies can be detected; jaywalking, pedestrian in danger, cyclist in danger, traffic entity too fast, traffic stopped, pedestrian loitering, vehicle against flow of traffic, etc. The current implementation supports the detection of 'street' and 'sidewalk' regions along with pedestrian in danger and jaywalking anomalies. The system learns in real-time and is capable of detecting anomalies after a cold start of few seconds of operation with no prior training. As well as the autonomous goal satisfying the system can also respond to user questions in real-time.

The effectiveness of learning mappings for question-answering has been demonstrated by [3], at least for simple domains. However, their approach requires $(image, question, answer)$ triples, essentially requiring questions to be provided at training time. This is not feasible in cases where novel questions are input by an operator at any time and require a real-time response. Also their model has no time-dependence and no way to make use of background knowledge, while the system we introduce has a notion of time and has the ability to use background knowledge, allowing it to identify user-relevant situations, and to make predictions about the future situation. The prediction capability allows the system to identify potential anomalies before they occur rather than simply detecting them after the event occurs.

2 Architecture

Multi-Class Multi-object Tracker is the main component of a distributed asynchronous real-time architecture used to produce highly scalable applications. The overall system is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events. In particular, a video streamer based on GStreamer video pipelines that receives multi-source streaming protocols (HLS, DASH, RTSP, etc.) and distributes video frames in compressed or raw formats, a multi-object Deep Neural network (DNN) detector based on YOLOv3 [8], and a multi-class tracker (MC-MOT) [7]. The outputs of the of MC-MOT together with the input video frames are eventually merged and sent in sync to the openNARS-based visual reasoner for further processing. Interprocess communication and data exchange within the different components is done using Redis® an in-memory data structure store configured as LRU cache for reliable online operations.

The tracking problem is about how to recognize individual objects in such a way, that they keep their ID while moving continuously in time. We used a tracking-by-detection approach for multi-objects (MOT) that has become increasingly popular in recent years, driven by the progress in object detection and convolutional neural network (CNN). These systems perform predictions (Kalman filter) and linear data association (Hungarian algorithm) linking multiple object detections (YOLOv3) belonging to the same class within a video sequence. The multi-class multi-object tracker extends this concept to tracking for objects belonging to multiple classes (MC-MOT) by forking a MOT for each class of interest that in this project we limit to three classes: Person, Bike, and Car. The CNN multi-object detector publishes the objects detection positions in pixel coordinates (x, y, w, h), while the three MOTs subscribe to a specific object Class and receives only the corresponding objects detection. The output of each MOT is represented by a segment of trajectory or tracklets, each having a class ID, instance ID, and a sequence of n previous detections $(x_1, y_1, t_1, w_1, h_1), ..., (x_n, y_n, t_n, w_n, h_n)$, where x_i, y_i represents the location in pixel units on the X- and Y-axis, t_i the timestamp of the detection and w_i, h_i the width and height of the corresponding object bounding box. On average, it takes 30ms for the objects detection on an NVIDIA® Tesla® P100 board, and 15ms for tracking. Thus, we can safely provide tracklets and frames at 15fps including system latency to the openNARS visual reasoner, that is fair enough for real-time applications.

OpenNARS is an implementation of Non-Axiomatic Reasoning System (NARS). NARS is a general-purpose reasoning system that works under the Assumption of Insufficient Knowledge and Resources. It operates using an inference cycle that applies all matching Non-Axiomatic Logic-based inference rules that apply to the selected premises (see [5]). The selection happens in a probabilistic way, following an Attention-based control strategy. The details of the memory and control mechanism are outside of the scope of the paper (see [1] for it), but the knowledge representation will be needed to understand the following sections.

Knowledge representation [5] As a reasoning system, NARS uses a formal language called "Narsese" for knowledge representation, which is defined by a formal grammar given in [5]. To fully specify and explain this language is beyond the scope of this article, so in the following only the directly relevant part is introduced informally and described briefly.

The logic used in NARS belongs to a tradition of logic called "term logic", where the smallest component of the representation language is a "term", and the simplest statement has a "subject-copula-predicate" format, where the subject and the predicate are both terms.

The basic form of statement in Narsese is *inheritance statement* which has a format "$S \rightarrow P$", where S is the subject term, and P is the predicate term, the "$\rightarrow$" is the *inheritance* copula, which is defined as a reflexive and transitive relation from one term to another term. The intuitive meaning of "$S \rightarrow P$" is "S is a special case of P" and "P is a general case of S". For example, statement "$robin \rightarrow bird$" intuitively means "Robin is a type of bird".

We define the *extension* of a given term T to contain all of its known special cases, and its *intension* to contain all of its known general cases. Therefore, "$S \rightarrow P$" is equivalent to "S is included in the extension of P", and "P is included in the intension of S".

[5] This subsection was adapted from [6] to make the paper self-contained.

The simplest, or "atomic", form of a term is a *word*, that is, a string of characters from a finite alphabet. In this article, typical terms are common English nouns like *bird* an *animal*, or mixed by English letters, digits 0 to 9, and a few special signs, such as hyphen('-') and underscore ('_'), but the system can also use other alphabets, or use terms that are meaningless to human beings, such as "drib" and "aminal".

Beside atomic terms, Narsese also includes *compound terms* of various types. A compound term $(con, C_1, C_2, ..., C_n)$ is formed by a term connector, *con*, and one or more component terms $(C_1, C_2, ..., C_n)$. The term connector is a logical constant with pre-defined meaning in the system. Major types of compound terms in Narsese includes:

- **Sets:** Term $\{Tom, Jerry\}$ is an *extensional set* specified by enumerating its instances; term $[small, yellow]$ is an *intensional set* specified by enumerating its properties.
- **Intersections and differences:** Term $(bird \cap swimmer)$ represents "birds that can swim"; term $(bird - swimmer)$ represents "birds that cannot swim".
- **Products and images:** The relation "John is the uncle of Zack" is represented as "$(\{John\} \times \{Zack\}) \to uncle\text{-}of$", "$\{John\} \to (uncle\text{-}of \,/\, \diamond\ \{Zack\})$", and "$\{Zack\} \to (uncle\text{-}of \,/\, \{John\} \diamond)$", equivalently.[6] Here, $\diamond$ is a placeholder which indicates the position in the *uncle-of* relation the subject term belongs to.
- **Statement:** "John knows soccer balls are round" can be represented as a *higher-order statement* "$\{John\} \to (know \,/\, \diamond\ \{soccer\text{-}ball \to [round]\})$", where the statement "$soccer\text{-}ball \to [round]$" is used as a term.
- **Compound statements:** Statements can be combined using term connectors for disjunction('$\vee$'), conjunction('$\wedge$'), and negation('$\neg$'), which are intuitively similar to those in propositional logic, but not defined using truth-tables.[7]

Several term connectors can be extended to take more than two component terms, and the connector is often written before the components rather than between them, such as $(\times \{John\}\, \{Zack\})$.

Beside the *inheritance* copula ('$\to$', "is a type of"), Narsese also has three other basic copulas: *similarity* ('$\leftrightarrow$', "is similar to"), *implication* ('$\Rightarrow$', "if-then"), and *equivalence* ('$\Leftrightarrow$', "if-and-only-if"), and the last two are used between statements.

In NARS, an *event* is a statement with temporal attributes. Based on their occurrence order, two events E_1 and E_2 may have one of the following basic temporal relations:

[6] This treatment is similar to the set-theoretic definition of "relation" as set of tuples, where it is also possible to define what is related to a given element in the relation as a set. For detailed discussions, see [5].

[7] The definitions of disjunction and conjunction in propositional logic do not require the components to be related in content, which lead to various issues under AIKR. In NARS, such a compound is formed only when the components are related semantically, temporally, or spatially. See [5] for details.

- E_1 happens before E_2
- E_1 happens after E_2
- E_1 happens when E_2 happen

More complicated temporal relations can be expressed by taking about the sub-events of the events.

Temporal statements are formed by combining the above basic temporal relations with the logical relations indicated by the term connectors and copulas. For example, implication statement "$E_1 \Rightarrow E_2$" has three temporal versions, corresponding to the above three temporal orders, respectively:[8]

- $E_1 \mathrel{/\!\!\Rightarrow} E_2$
- $E_1 \mathrel{\backslash\!\!\Rightarrow} E_2$
- $E_1 \mathrel{|\!\!\Rightarrow} E_2$

Conjunction statement "$E_1 \wedge E_2$" has two temporal versions, corresponding to two of the above three temporal orders, respectively:

- (E_1, E_2) (forward)
- $(E_1; E_2)$ (parallel)

All the previous statements can be seen as Narsese describing things or events from a third-person view. Narsese can also describe the actions of the system *itself* with a special kind of event called *operation*. An operation is an event directly realizable by the system itself via executing the associated code or command.

Formally, an operation is an application of an operator on a list of arguments, written as $op(a_1, \ldots, a_n)$ where op is the operator, and $a_1, ..., a_n$ is a list of arguments. Such an operation is interpreted logically as statement "$(\times \{SELF\} \{a_1\} \ldots \{a_n\}) \rightarrow op$", where $SELF$ is a special term indicating the system itself, and op is an operator that has a procedural interpretation. For instance, if we want to describe an event "The system is holding key_001", the statement can be expressed as "$(\times\{SELF\} \{key\_001\}) \rightarrow hold$".

Overall, there are three types of sentences defined in Narsese:

- A **judgment** is a statement with a truth-value, and represents a piece of new knowledge that system needs to learn or consider. For example, "$robin \rightarrow bird \ \langle f, c\rangle$", where the truth-value $\langle f, c\rangle$ will be introduced in the next section.
- A **question:** is a statement without a truth-value, and represents a question to be answered according to the system's beliefs. For example, if the system has a belief "$robin \rightarrow bird$" (with a truth-value), it can be used to answer question "$robin \rightarrow bird?$" by reporting the truth-value, as well as to answer the question "$robin \rightarrow ?$" by reporting the truth-value together with the

[8] Here the direction of the arrowhead is the direction of the implication relation, while the direction of the slash is the direction of the temporal order. In principle, copulas like '$/\!\!\Leftarrow$' can also be defined, but they will be redundant. For more discussion on this topic, see [5].

term *bird*, as it is in the intension of *robin*. Similarly, the same belief can also be used to answer question "? $\rightarrow$ *bird*" by reporting the truth-value together with the term *robin*.
- A **goal** is statement without a truth-value, and represents a statement to be realized by executing some operations, according to the system's beliefs. For example, "$(\times \{SELF\} \{door\_001\}) \rightarrow open!$" means the system has the goal to open the *door*_001 or to make sure that *door*_001 is opened. Each goal has a "desire-value", indicating the extent to which the system hopes for a situation where the statement is true.

Tracklets to Narsese To map tracklets to NARS events, the numeric information encoded in each tracklet is discretized. This can happen in many ways. We used a fixed-sized grid that maps every detection tuple to the rectangle it belongs to, as shown in Figure 1:

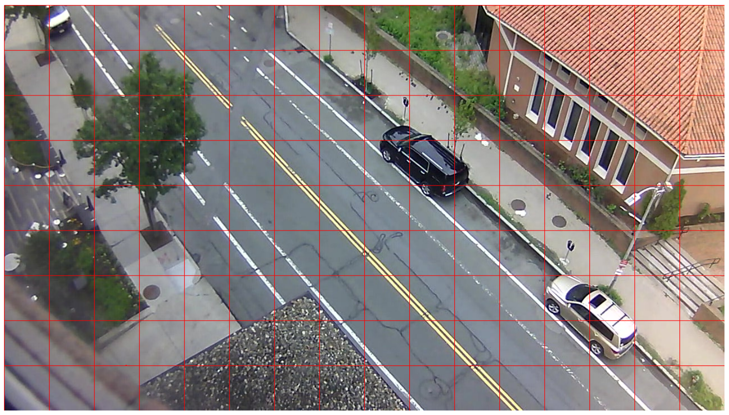

Fig. 1: Spatial discretization by a grid

Also for each detection, a *Angle* term is built based on a discretization of the overall tracklet direction, which can be

- *11* = left up
- *10* = left down
- *01* = right up
- *00* = right down
  (To be expanded to 8 compass points)

Additionally the class and instance ID is provided, which (currently) can be *Car*, *Cycle* or *Pedestrian*. Now using the above, the following events can be built:

- Indicating the class of an instance:
  $\{InstanceID\} \rightarrow Class$
- Indicating the direction of an instance:
  $(\{InstanceID\} \times \{Angle\}) \rightarrow directed$
- Indicating the position of an instance:
  $(\{InstanceID\} \times \{RectangleID\}) \rightarrow positioned$

– To reduce the amount of input events, also combinations are possible: $(\{InstanceID\} \times \{RectangleID\} \times \{Angle\} \times \{Class\}) \rightarrow Tracklet$
– Additionally, the system learns to assign *street* and *sidewalk* labels to the scene, based on the car and pedestrian activity. This is achieved through the use of an implication statement $((\{\#1\} \rightarrow Car); (\{\#1\} \times \{\$2\}) \rightarrow positioned) \not\Rightarrow (\{\$2\} \rightarrow street)$. and $((\{\#1\} \rightarrow Pedestrian); (\{\#1\} \times \{\$2\}) \rightarrow positioned) \not\Rightarrow (\{\$2\} \rightarrow sidewalk)$. Whenever the consequence is derived, it will be revised up, and the choice rule will select the candidate of highest truth expectation (either *street* or *sidewalk*) to answer the question $\{specificPosition\} \rightarrow ?X$
– In addition, relative location relations R are provided by the system, including $leftOf$, $rightOf$, $topOf$, $belowOf$ and $closeTo$. These are encoded by $(\{InstanceID1\} \times \{InstanceID2\}) \rightarrow R$
Please note that the closeTo relation is only input when the distance is smaller than some threshold defined by the system operator.

3 Smart City Results

Street Scene (see [4]) is a dataset for anomaly detection in video data. It is data collected from a camera mounted on a building, watching a street. The dataset includes unusual cases that should be detected, such as jaywalking pedestrians, cars driving on the sidewalk, or other atypical situations. The tracker applied to the video dataset, outputs tracklets as introduced previously, which are then encoded into Narsese as described. NARS then can use the input information to make predictions, satisfy goals, or to answer queries in real time. Also NARS can detect anomalies and classify them with a background ontology.

Initially we developed a street scene simulator, **Crossing**, simulating a real street. This allowed us to simulate the traffic and pedestrians on a street and generate situations/anomalies that we did not have real data for. We then tested our system on the following tasks:

Prediction First we tested OpenNARS's prediction ability with the Crossing simulator (see [2], and Figure 3).

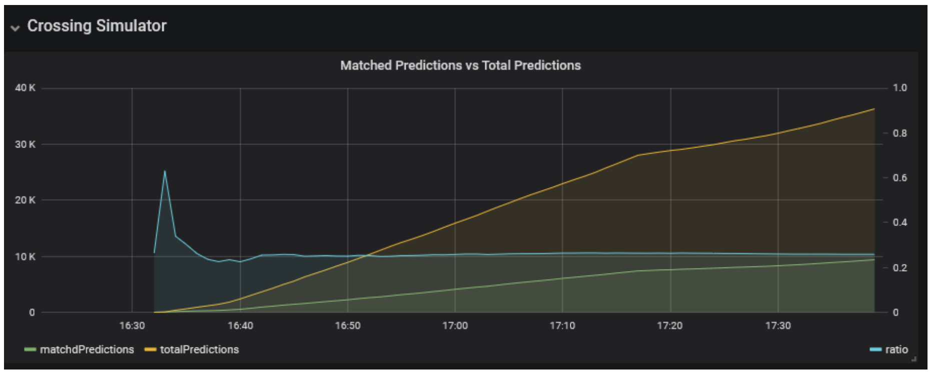

Fig. 2: Prediction performance in the Crossing simulator

After initial fluctuation (see Figure 2) due to small sample size, after a short time, the system reaches a stable prediction success ratio, while the total amount of predictions made increases. While the overall tendency is clear, the exact convergence ratio depends on the penalties given to differences between predicted and occurred location, where less penalty results in higher success percentage. These results are initial, and show the system's stability. Setting a higher truth expectation threshold increases the prediction accuracy, but in the future the percentages will be improved for a fixed threshold.

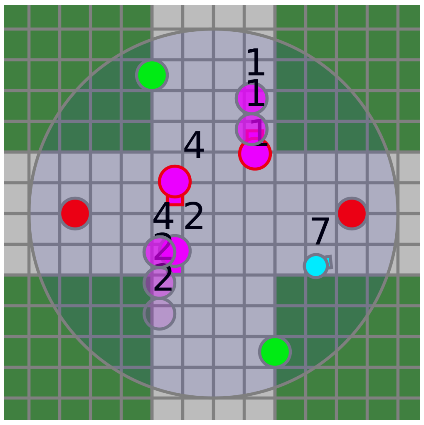

Fig. 3: Car predictions in the Crossing simulator shown in magenta

Reasoning-based annotation In the Street Scene dataset, our system is able to label a location as street based on the tracklet activity of pedestrians and cars. This usually happens in an one-shot way, but will be overridden or strengthened by further evidence.

Question Answering Also in Street Scene, we tested the system's ability to answer questions about the current situation, in real time, demonstrating situational awareness about the current state of the street. Questions included $((\{?1\} \rightarrow Class);(\{?1\} \times \{LocationLabel\}) \rightarrow located)?$ where ?1 is a variable queried for, essentially asking for an instance of a specific class at a specific location such as lane1, which the system returns immediately when perceived, allowing a user, for instance, to ask for jaywalking pedestrians.

Anomaly Detection Often a system should not operate passively (answer queries), but automatically inform the user when specific cases occur. Our approach allows the usage of background knowledge to classify unusual situations, and to drop the user a message, if desired.

For instance, consider the case of a moving car getting too close to a pedestrian, putting the person in danger. This can easily be expressed in Narsese, using the previously mentioned relative spatial relations the system receives. Furthermore, it can be linked to a goal, such that the system will inform the user whenever it happens: $(((\{\#1\} \rightarrow Car);((\{\#2\} \rightarrow Pedestrian))(\{\#1\} \times \{\#2\}) \rightarrow closeTo), say(\#2, is\_in\_danger)) \mathrel{/\!\!\Rightarrow} (\{SELF\} \rightarrow [informative])$. which will let the system inform the user assuming the goal $(\{SELF\} \to [informative])!$ was given to the system. An example can be seen in Figure 4.

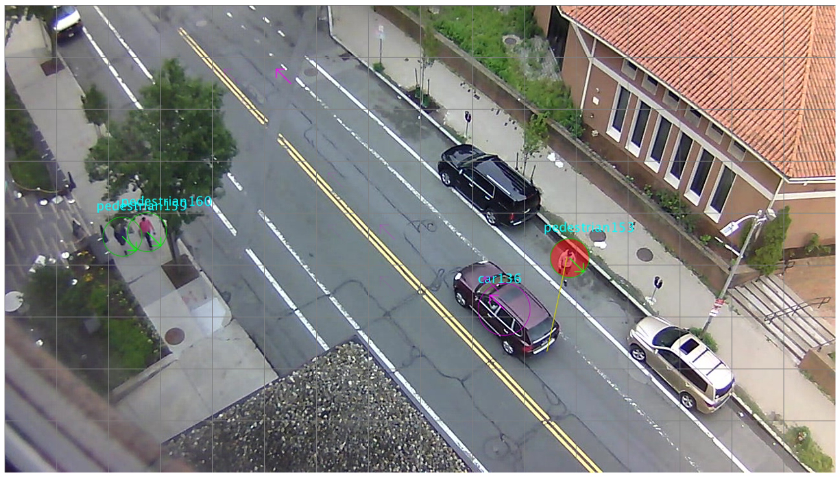

Fig. 4: A pedestrian in danger due to close proximity to a moving car

Also jaywalking pedestrians can be specified similarly, using $(((\{\#1\} \to Pedestrian)); (\{\#1\} \times \{street\}) \to at)), say(\#2, is\_jaywalking))$ $\not\Rightarrow (\{SELF\} \to [informative])$. An example can be seen in Figure 5.

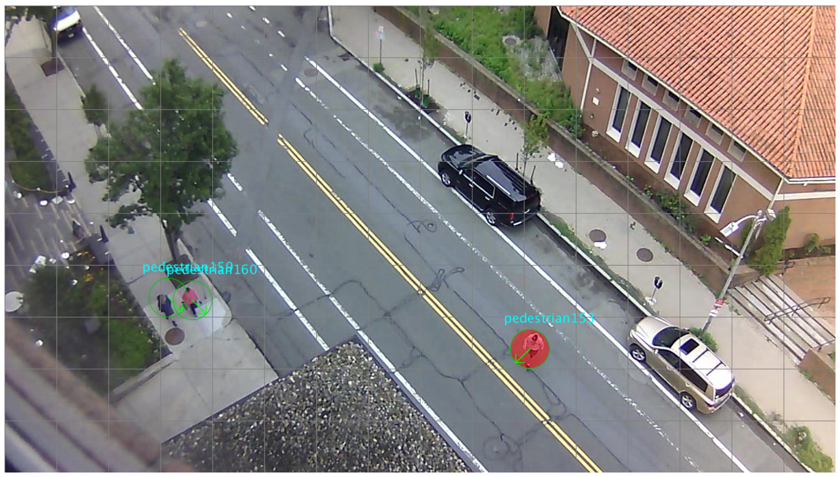

Fig. 5: An example of Jaywalking

In our tests, the system had no issue identifying these cases whenever they occurred through all the examples existing in the Street Scene dataset.

4 Conclusion

Our system demonstrates predictive capabilities using Non-Axiomatic Reasoning, based on tracklet representations of objects provided by the multi-class multi-object tracker (MC-MOT), which are converted to Narsese as we described.

Also, the relative and absolute location information given to the system, together with object instances, object categories and other attributes, allows for a rich set of questions to be asked and answered by the system. Here, the system has proven to be able to label streets and sidewalks automatically, and has shown to be capable of answering conjunctive queries with variables in real time, while the scene is changing, and to detect anomalies using a simple ontology.

Additionally, we have shown the ability to let the system work autonomously, informing the user in situations of interest, guided by background knowledge and a goal driven system to inform the user. Future work will include extending the system to support the full anomaly ontology mentioned previously along with a wider range of tracked entity classes, along with improvements to the prediction accuracy.

References

1. Hammer, P., Lofthouse, T., & Wang, P. (2016, July). The OpenNARS implementation of the non-axiomatic reasoning system. In International conference on artificial general intelligence (pp. 160-170). Springer, Cham.
2. OpenNARS applications https://github.com/opennars/opennars-applications, last accessed: June 25, 2019.
3. Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. arXiv preprint arXiv:1904.12584.
4. Ramachandra, B., & Jones, M. (2019). Street Scene: A new dataset and evaluation protocol for video anomaly detection. arXiv preprint arXiv:1902.05872.
5. Wang, P. (2013). Non-axiomatic logic: A model of intelligent reasoning. World Scientific.
6. Wang, P., Li, X., & Hammer, P. (2018). Self in NARS, an AGI System. Frontiers in Robotics and AI, 5, 20.
7. KangUn Jo, JungHyuk Im, Jingu Kim and Dae-Shik Kim (2017). A real-time multi-class multi-object tracker using YOLOv2. International Conference on Signal and Image Processing Applications, 2017, Kuching, Malaysia.
8. J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.