Model Assertions for Debugging Machine Learning

Daniel Kang*, **Deepti Raghavan***, **Peter Bailis, Matei Zaharia**
Stanford DAWN Project

Abstract

Machine learning models are increasingly being deployed in mission-critical settings, such as self-driving cars. However, these models can fail in complex ways (e.g., a vehicle on autopilot repeatedly accelerated towards lane dividers), so it is imperative that application developers find ways to debug these models. We propose adapting software assertions, or boolean statements about the state of a program that must be true, to the task of debugging ML models. With *model assertions*, ML developers can specify constraints on model outputs, e.g., cars should not disappear and reappear in successive frames of a video. Model assertions can be exact or "soft," i.e., probabilistic. We propose several ways to use model assertions in ML debugging, including use in runtime monitoring, in performing corrective actions, and in collecting "hard examples" to further train models with human labeling or weak supervision. We show that, for a video analytics task, simple assertions can effectively find errors and correction rules can effectively correct model output (up to 100% and 90% respectively). We additionally collect and label parts of video where assertions fire (as a form of active learning) and show that this procedure can improve model performance by up to $2\times$.

1 Introduction

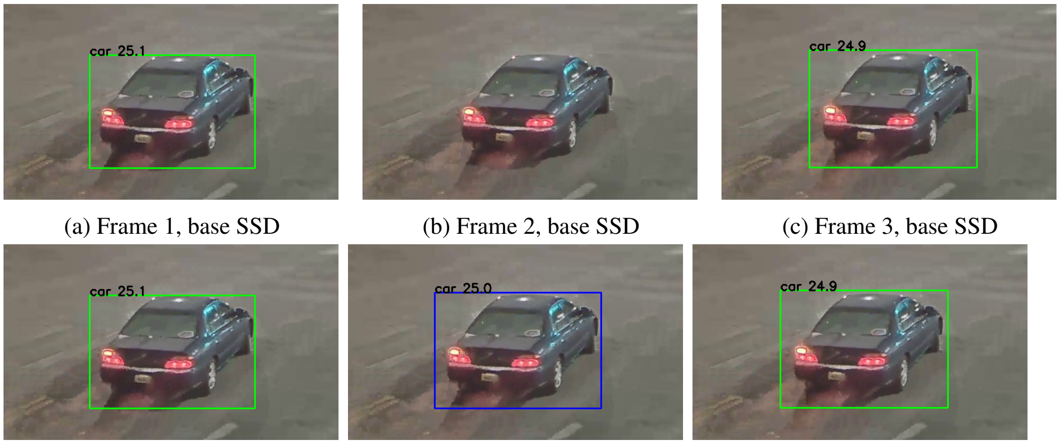
ML is increasingly used in mission-critical contexts, such as in self-driving vehicles [1] or in setting bail [2]. However, ML models can exhibit unpredictable behavior on real world-tasks. For example, Tesla's cars suffered multiple incidents where they accelerated into one type of highway lane divider [3] and Google's autonomous vehicle collided with a bus because the car expected the bus to yield, but the bus did not [4]. Thus, it is critical to be able to debug ML models and applications. Unfortunately, there is currently no standard means of debugging ML models.

In this work, we investigate the potential to apply one of the most basic techniques in software quality assurance—assertions [5, 6]—to debug and improve ML models. Software engineers have built software for a wide range of mission-critical settings, such as spaceships and medical devices, for decades using a variety of quality assurance and error detection techniques. Program assertions, or boolean statements that must be true at execution time (e.g., the length of an array must be greater than zero), are used as the "first line of defense" in software and have been shown to significantly reduce the number of bugs [7].

We explore several means of using *model assertions*, assertions applied to the outputs of ML models, to debug and improve models. We consider both "exact assertions," deterministic functions on model outputs that are similar to traditional program assertions, and "soft assertions," which have high, but not perfect, precision. For example, when identifying and localizing objects in video for an object detection task, detected objects can "flicker" in and out between consecutive frames (Figure 1), or can be highly overlapped or nested (Figure 2). An assertion over model outputs could easily detect both

*Equal Contribution

Preprint. Work in progress.



(a) Frame 1, base SSD (b) Frame 2, base SSD (c) Frame 3, base SSD

(d) Frame 1, base SSD (e) Frame 2, fixed frame (f) Frame 3, base SSD

Figure 1: **Top row:** example of flickering in three consecutive frames of a video. The object detection method failed to identify the car in the second frame. **Bottom row:** example of correcting the output of a model. The car bounding box in the second frame can be automatically filled in using the frames around it. Best viewed in color.

these errors. It may be useful to make these soft assertions because some rare real-world situations do have flickering and nested objects (e.g., a video advertisement on the side of a bus).

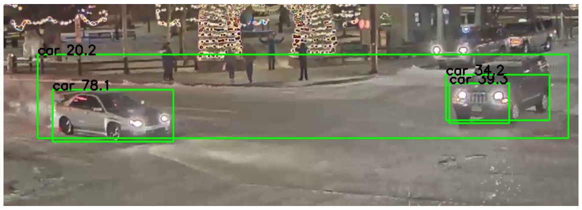
We explore several ways to use model assertions, at runtime and at training time:

- **Runtime monitoring**: At runtime, model assertions can be used to collect statistics on incorrect behavior.
- **Corrective action**: At runtime, model assertions can be used to trigger corrective action, e.g., by returning control to a human operator.
- **Active learning**: At training time, model assertions can be used to decide which new data points to request human-annotated labels for, for active learning [8]. As model assertions are exact or high-precision, the output of the model will likely be wrong when the assertion triggers. Collecting data that is "difficult" for models is known to be a viable training data collection strategy [9].
- **Weak supervision**: In some cases, analysts can write automatic *corrective rules* to propose new labels when an assertion fires, which can then be used as a form of weak supervision [10, 11] to retrain the model on data where the assertion failed. For example, for the flickering assertion, a corrective rule might fill in labels from adjacent frames.

To investigate the potential of model assertions, we evaluate a prototype implementation on a video analytics task. Specifically, we consider the task of object detection (the task of localizing and classifying objects) for monitoring applications. We use a single-shot detector (SSD) [12] to determine the location of cars in a street intersection and find that it performs poorly (40.4% mAP). As a result, we implement two assertions over the output of the detector. We demonstrate that these simple assertions can be written with near 100% precision. We additionally collect and label frames of video where the assertions were triggered, which we then use to further train SSD as a form of active learning. We show this procedure can result in up $2\times$ improvement mAP (82.6%). Finally, we show that weak supervision using correction rules can improve mAP by 5% with no human labeling.

2 Related Work

Verified Machine Learning. We target complex, real-world scenarios, where complete specifications may not be possible. In limited scenarios, there has been an increasing line of work for verifying machine learning models. For example, Reluplex [13] can verify that extremely small networks will make correct control decisions given a fixed set of inputs and other work has shown that similarly small networks can be verified against minimal perturbations of a fixed set of input images [14].



(a) Example 1.

(b) Example 2.

Figure 2: Examples of when the multi-box assertion fires. Best viewed in color.

```
for i in range(cur_frame - 1, cur_frame - 10):
  similar_boxes = get_similar_cars(cur_boxes, past_boxes[i])
  if len(similar_cars) == 0: # no similar boxes
    for j in range(i - 1, cur_frame - 10):
      overlapping_boxes = get_similar_cars(cur_boxes, past_boxes[j])
      if len(overlapping_boxes) == 0:
        continue
      else:
        raise FlickerException
  else:
    cur_boxes = similar_boxes
```

Figure 3: The pseudo-code for the flickering assertion. For each frame, the assertion inspects recent frames and fires if there are any "gaps" between frames, e.g., if the model identifies a car in frame *i* and *i - 2*, but not *i - 1*.

However, these verifications are done for very limited scenarios (e.g., small L_∞ perturbations of images) and not the complex, real-world scenarios we target.

Structured Prediction, Inductive Bias. Several ML methods encode structure or inductive biases into the training procedure or models themselves [15, 16]. For example, structured prediction attempts to predict output with additional constraints (e.g., object detection) [15]. While promising, designing algorithms and models with specific inductive biases can be challenging for non-experts. Additionally, these methods generally do not contain runtime checks for aberrant behavior.

Software Debugging. Writing correct software and verifying the correctness of software has a long history, with many proposals from the research community. We hope that many such practices are adopted in deploying machine learning models; we focus on assertions in this work [5, 6]. Assertions have been shown to reduce the prevalence of bugs, when deployed correctly [7, 17]. There are many other such methods, such as formal verification [18, 19, 20], methods of conducting large-scale testing (e.g., fuzzing) [21, 22], and symbolic execution to trigger assertions [23, 24].

Weak Supervision, Semi-supervised Learning. The goal of weak supervision is to leverage higher-level and/or noisier input from human experts to improve the quality of models [25, 10, 11]. In semi-supervised learning, structural assumptions over the data are used to leverage unlabeled data [26]. These methods are generally used to expand the training data. For example, human experts can write *labeling functions* to weakly annotate data, which can be used as training data to improve the performance of models [11, 27]. Flipper [28] explores debugging the quality of automatically generated training data. However, these methods generally do not contain runtime checks and, to the best of our knowledge, have not been used as a form of active learning.

3 Overview

In this section, we give an example of how model assertions can be used and implemented.

Suppose an analyst were studying traffic flow through an intersection and deploys an object detection method [29]. The analyst notices that the object detection method predicts objects in some frames, but occasionally does not detect an object as shown in Figure 1 (i.e., flickering).

The assertion can be written as in Figure 3, in which the assertion triggers if there are "gaps" of boxes between frames. Additionally, to correct the model behavior, the missing boxes can be filled in from the by interpolating from the boxes in the previous frames.

We explore the how model assertions can be applied to this setting in four ways, for both real-time and batch analytics senarios. For monitoring, the analyst can collect statistics on the number of errors to detect model drift. For corrective action, the analyst can manually inspect the data. For active learning, the frames where the assertion fired could be stored and later be human-annotated. For weak supervision, the analyst can write a corrective rule that can generate weak labels to further train the model.

4 Methods

We implement a prototype for model assertions called OMG (we envision OMG to be a "model guardian"; OMG is a recursive acronym for OMG Model Guardian). We evaluate OMG on video analytics and plan to apply model assertions to other domains.

OMG consists of three components: an API for specifying assertions, a runtime engine for detecting assertions when models are deployed, and a pipeline for improving models by using assertions with weak supervision or active learning. We describe each component in turn.

OMG currently implements assertions as user defined functions (UDFs) over the model output. For video analytics, model assertions accept the output of the model at the current frame and some limited history. For example, for object detection, the assertion would use the bounding boxes from the current frame and (for example) the past 10 frames as input. While this evaluation targets object detection, we could also apply model assertions to semantic segmentation [30], instance segmentation [31], or image classification [32]. As with program assertions, model assertions can perform arbitrary computations. Finally, model assertions can have corrective rules associated with them, which can generate weak labels for further retraining (e.g., filling in boxes for flickering).

At runtime, one or more assertions are run over the output of the current frame, along with a short history of outputs from previous frames. At runtime, OMG can take several actions if an assertion is triggered. First, OMG can record statistics which can subsequently be analyzed (e.g., to detect model drift). Second, OMG can perform corrective action by returning control to a human operator. Finally, OMG can store the data that caused assertions for further processing, such as manual analysis or active learning.

Finally, data collected from when the assertions fired at runtime can be used as training data to further improve the model. As model assertions are ideally high precision (i.e., the output is highly likely to be wrong when the assertion is triggered), the output of the model is likely to be incorrect. Collecting instances of training data that are "hard" for models is known to to be a viable strategy for improving model quality [9]. Data that fails assertions can be labeled in one of two ways. First, this data can be sent to human annotators as a form of active learning. Second, if the model output can be corrected automatically, analysts can write corrective rules that propose new labels for the data on as a form of weak supervision [10, 11].

5 Experiments

We evaluate OMG on a real-world security camera to see 1) if model assertions can effectively find errors and 2) if model assertions can improve model performance. We find that simple model assertions can be written to be highly precise and can be used to significantly improve model performance.

5.1 Experimental Configuration and Model Assertions

Dataset and model. We evaluate OMG on the `night-street` video from [33]. We use post processed Mask-RCNN labels from [33] as ground truth, with a different day for training and testing. The production model we debug and improve is SSD [12], a much faster object detection method

```
counter = [0 for i in range(len(boxes))]
for i in range(len(boxes)):
  for j in range(len(boxes)):
    if i == j:
      continue
    if nearly_contains(box[i], box[j]):
      counter[i] += 1
      if counter[i] >= 2:
        raise MultiBoxAssertion
```

Figure 4: The pseudo-code for the multibox assertion. If a box nearly contains at least two other boxes, the assertion fires.

Assertion	% of frames with incorrect behavior	% of frames fixed correctly
Flickering	100%	90%
Multibox	100%	N/A

Table 1: Assertions and the percentage of time that they correctly identify an incorrect behavior and percentage of time correction rules correctly fix the output. The multibox assertion does not correct the output.

than Mask R-CNN. Single-shot detectors are widely used in practice [34, 35, 36]. SSD executes 20× faster than Mask R-CNN, but produces more errors on our test video. Thus, we evaluate whether we can find and reduce these errors using model assertions.

Assertions. We implemented two assertions: 1) to detect flickering in video ("flickering") and 2) to detect if a box of a car contains two other boxes of cars ("multibox"). An example of flickering is shown in Figure 1; while methods have been proposed to reduce flickering in video [37, 38], OMG can use model assertions both for monitoring runtime behavior and for active learning or weak supervision to improve SSD's performance. We also wrote a corrective rule for flickering that automatically fills in labels based on the nearby boxes in adjacent frames for our weak supervision experiments. An example of multibox is shown in Figure 2; the pseudo-code is shown in Figure 4. We were unable to construct a high-precision corrective rule for the multibox assertion.

5.2 Model Assertions can Identify a Large Fraction of Errors

We ran inference using SSD over nine hours of the `night-street` video and then ran the flickering and multibox model assertions over the output to see if assertions can effectively find errors. For each assertion, we manually annotated 50 random frames where the assertion fired to check whether the assertion actually caught a real mistake. For flickering, we also verified whether the correction rule produced a reasonable label. The results are shown in Table 1, the assertion correctly identifies an issue 100% of the time and, in the case of flickering, the correction rule correctly fixes the output 90% of the time. Thus, we see that simple assertions both can be highly precise and can effectively correct model output.

5.3 Model Assertions can Improve Model Performance

To see if using model assertions as a form of active learning or weak supervision can improve model performance, we collect frames where each assertion fired and random frame to use as training data, from 9 hours of video. We finetuned the following variants of SSD, pretrained on MS-COCO with 2000 frames each:

- SSD trained with 1000 frames that triggered the flickering assertion and 1000 random frames, labeled via weak supervision.
- SSD trained with 2000 random frames labeled via active learning, as a baseline.
- SSD trained with 1000 frames that triggered the flickering assertion and 1000 random frames, labeled via active learning.
- SSD trained with 600 frames that triggered the multibox assertion and 1400 random frames, labeled via active learning.
- SSD trained with 1400 frames that triggered the flickering assertion and 600 frames that triggered the multibox assertion, labeled via active learning.

Model	mAP	Recall at ~80% precision
Regular SSD	33.2	36.4%
SSD, weak supervision (flickering)	49.1	62.2%
SSD, active learning (random)	66.0	82.4%
SSD, active learning (multibox)	67.2	85.9%
SSD, active learning (flickering)	68.9	85.9%
SSD, active learning (flickering + multibox)	70.5	87.7%

Table 2: Performance of the standard SSD, SSD trained with weak supervision for flickering, and SSD trained with active learning on unseen data for flickering. As shown, both weak supervision and active learning improve SSD. Assertion-based active learning outperforms labeling random frames at a fixed labeling budget.

Model	Data	% of labels with flickering	Reduction
Regular SSD	Seen	14.3%	1×
SSD, weak supervision	Seen	3.8%	3.8×
SSD, random frames	Seen	2.5%	5.7×
SSD, active learning	Seen	1.8%	7.8×
Regular SSD	Unseen	16.3%	1×
SSD, weak supervision	Unseen	3.4%	4.8×
SSD, random frames	Unseen	2.5%	6.4×
SSD, active learning	Unseen	2.2%	7.5×

Table 3: Number of frames where flickering occurred for the standard SSD, SSD trained with weak supervision, and SSD trained with active learning (using both assertions), both on the data that generated the supervision and unseen data. As shown, the number of frames with flickering is reduced after training and assertion-based active learning outperform random labeling.

As we were unable to find a correction rule for the multibox assertion, we only ran weak supervision for the flickering assertion. We compare to an SSD pretrained on MS-COCO. In each experiment, we used a learning rate of 5×10^{-5} (active learning) or 10^{-5} (weak supervision) and ran training for 6 epochs. We averaged two runs of retraining.

Our accuracy metrics include mean average precision (mAP), a standard metric used in object detection [39], and recall at ~80% precision (which is what SSD achieves at a reasonable confidence threshold relative to Mask R-CNN). We additionally count the number of times flickering occurred before and after training. For testing, we run inference on one hour of video footage from a different day than training.

As shown in Table 2, we see that both weak supervision and active learning improve the model's performance. As expected, active learning improves the model significantly (as it uses ground truth labels) and outperforms sampling random frames at a fixed labeling budget of 2000 frames. We additionally show the number of frames flickering occurred before and after training in Table 3. As shown, the number of frames where flickering occurred is significantly reduced after training with both weak supervision and active learning.

6 Conclusion

As machine learning models continue to be deployed in mission-critical settings, the need for principled approaches of debugging machine learning models only increases. In this work, we propose adapting program assertions to machine learning models, through the concept of model assertions. We implement a prototype to deploy model assertions and apply it to video analytics. We demonstrate that simple assertions can catch many errors and that they can be used as a form of weak supervision and active learning to significantly improve model accuracy. Moving forward, we plan to evaluate model assertions in more domains. Additionally, we hope to see other practices from software engineering (e.g., large-scale fuzzing) adapted to machine learning applications.

Acknowledgments

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Intel, Microsoft, NEC, Teradata, SAP, and VMware—as well as DARPA under No. FA8750-17-2-0095 (D3M), industrial gifts and support from Toyota Research Institute, Keysight Technologies, Hitachi, Northrop Grumman, NetApp, and the NSF under grants DGE-1656518 and CNS-1651570.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Avery Thompson. An algorithm is now helping set bail in new jersey, 2017.

[3] Timothy Lee. Tesla says autopilot was active during fatal crash in mountain view. `https://arstechnica.com/cars/2018/03/tesla-says-autopilot-was-active-during-fatal-crash-in-mountain-view/`, 2018.

[4] Chris Ziegler. A google self-driving car caused a crash for the first time. `https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report`, 2016.

[5] Herman Heine Goldstine, John Von Neumann, and John Von Neumann. Planning and coding of problems for an electronic computing instrument. 1947.

[6] Alan Turing. Checking a large routine. In *The early British computer conferences*, pages 70–72. MIT Press, 1989.

[7] Gunnar Kudrjavets, Nachiappan Nagappan, and Thomas Ball. Assessing the relationship between software assertions and faults: An empirical investigation. In *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pages 204–212. IEEE, 2006.

[8] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[9] Anonymous. Select via proxy: Efficient data selection for training deep networks. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[10] Alex Ratner, Stephen Bach, Paroma Varma, and Chris Ré. Weak supervision: The new programming paradigm for machine learning, 2017.

[11] Paroma Varma and Christopher Ré. Reef: Automating weak supervision to label training data. 2018.

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

[13] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[14] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[15] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, Ben Taskar, and SVN Vishwanathan. *Predicting structured data*. MIT press, 2007.

[16] David Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial intelligence*, 36(2):177–221, 1988.

[17] Aamer Mahmood, Dorothy M Andrews, and Edward J McCluskey. Executable assertions and flight software. 1984.

[18] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[19] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[20] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[21] Ari Takanen, Jared D Demott, and Charles Miller. *Fuzzing for software security testing and quality assurance*. Artech House, 2008.

[22] Patrice Godefroid, Michael Y Levin, and David Molnar. Sage: whitebox fuzzing for security testing. *Queue*, 10(1):20, 2012.

[23] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[24] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224, 2008.

[25] Paroma Varma, Bryan He, Dan Iter, Peng Xu, Rose Yu, Christopher De Sa, and Christopher Ré. Socratic learning: Augmenting generative models to incorporate latent subsets in training data. *arXiv preprint arXiv:1610.08123*, 2016.

[26] Xiaojin Zhu. Semi-supervised learning. In *Encyclopedia of machine learning*, pages 892–897. Springer, 2011.

[27] Alexander J Ratner, Stephen H Bach, Henry R Ehrenberg, and Chris Ré. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1683–1686. ACM, 2017.

[28] Paroma Varma, Dan Iter, Christopher De Sa, and Christopher Ré. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, page 5. ACM, 2017.

[29] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[31] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[33] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Fast exploratory video queries using neural networks. *arXiv preprint arXiv:1805.01046*, 2018.

[34] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[35] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[36] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[37] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016.

[38] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 3, 2017.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.