CLAIRE: Enabling Continual Learning for Real-time Autonomous Driving with a Dual-head Architecture

Hao Zhang
Dept. of Computer Science
North Carolina State University
Raleigh, NC, United States
hzhang47@ncsu.edu

Frank Mueller
Dept. of Computer Science
North Carolina State University
Raleigh, NC, United States
mueller@cs.ncsu.edu

***Abstract*—Autonomous vehicles rely on a pre-trained object detector to perceive surroundings. However, when never seen before scenarios are encountered, late decisions may result in hard braking due to perceived threats. Image sequences leading to such a situation provide the potential to learn and improve over time. Yet instant re-training on board with all prior training data is infeasible given computational, storage and power constraints. What's more, exposure of a pre-trained CNN to only images of the new scenario is known to result in "catastrophic forgetting" for already learned features.**

This work makes several contributions: A novel lightweight dual-head detection network architecture is proposed to overcome forgetting and to support fast on-board continual learning on small sets of new images and assesses the feasibility of continual learning methods for autonomous driving. A sensitivity study on the quality and quantity of continually learned images for our dual-head technique is performed, including an assessment of its real-time suitability. Experiments show that our method's accuracy is improved by up to 13% and performance increases by 5.8X over a state-of-the-art continual learning framework. This makes it suitable for autonomous driving scenarios with real-time constraints. Source code is made available via Github.

***Index Terms*—Autonomous Systems, On-board Continual Learning, Real-Time Deep Learning Inference**

I. Introduction

Automobiles are increasingly becoming complex computerized control systems consolidated into electronic control units (ECUs). This evolution is also driven by green energy trends replacing combustion engines with electric motors. Today, major technology companies, including Intel, Nvidia, Huawei, Baidu, Amazon, and Alphabet (Google), are pushing into the world of autonomous driving, where they find themselves competing with the "old" automotive giants, newcomers (Tesla), and automotive suppliers (Bosch, ZF, and Magna). In this setting, traditional control software is increasingly replaced by machine learning as control tasks are becoming more complex.

Many vendors are promoting their own on-board automotive hardware and software packages, often with customized operating systems (OS). Intel provides Mobileye [1], Nvidia offers its Drive OS [2], Volkswagen and Daimler have announced an OS [3] while Tesla has already deployed theirs [4], whereas Google promises Android Auto OS [5], which Ford and GM plan to use in next generation models. Many of these on-board auto OSs (including Nvidia's/Telsa's) are based on customized Linux versions, augmented with real-time kernels via virtualization (e.g., Autosar Adaptive Platform), delivering security, capability, and performance for drivers and passengers for higher-level control functionality. This trend to replace simple hardware control of traditional vehicles with more complex software systems provides new capabilities for autonomous driving and flexibly through over-the-air (OTA) updates. And while software updates have the potential to improve capabilities over time, making vehicles safer, smarter, and more energy efficient, these trends pose significant technical challenges. To realize vehicular autonomy, today's prototypes combine subsystems for perception, localization, prediction, planning, and control. In a perception subsystem, machine learning (ML) is widely employed. Camera/lidar images are subject to object detection and tracking based on CNNs. To decide on steering, acceleration and braking actions, object detection has to be accurate and fast, i.e., subject to real-time constraints.

YOLO [6], [7] is a state-of-the-art object detection framework based on one-stage CNN architecture that can meet real-time constraints (30 FPS). In contrast, two-stage architectures, such as R-CNN [8], Fast R-CNN [9] and Faster R-CNN [10], are more accurate but cannot meet real-time constraints as it suffers from much slower detection speed than one-stage detectors. The follow-up one-stage CNN architecture conquers two-stage ones in both accuracy and speed [11]. E.g., the inference speed of Faster R-CNN is 9.4 FPS, while one-stage Yolo achieves an inference speed of 54 FPS on the same hardware with a 1.4% improvement in accuracy [11]. To reach such frame rates, autonomous driving systems (ADS) feature hardware acceleration via embedded GPUs and FPGAs, but with lower performance than discrete acceleration devices. For these reasons, most autonomous driving systems (Tesla Autopilot, Baidu Apollo, etc.) leverage a one-stage detector in their perception system [6], [11], [12], as does our work.

Our work breaks new ground as we hypothesize that an

978-1-6654-0627-7/22/$31.00 ©2022 IEEE

on-board ADS should provide the ability to learn from "near-mistakes" with current hardware capabilities. Existing CNNs are far from perfect due to a bias in learning limited to the training set of images. If such a set lacks certain images, safety can be compromised. E.g., the image detector fails to detect a stop sign at first, but the secondary radar-based crash prevention system forced the vehicle to stop before hitting an object. If the system has the ability of reviewing image frames prior to the hard braking action, it would be able to detect the similar scenario next time. The concept of the automated online learning is based on rewinding a sequence of images (say, from a near impact where objects were identified late) and labeling the same, smaller objects of previous frames by tracking objects backwards. This labeling process can be unsupervised and is subject to ongoing work. (Another example would be a stop sign obscured by parked vehicles from a distance seen late, but smaller intersection indications including lines on the ground or the backside of a stop sign on the opposite lane could have alerted the system earlier. Rewinding and auto-labeling can help here as well.)

A simple solution would be to re-train a CNN with additional images plus all seen before images to improve detection capabilities, yet this would require days of computational training, e.g., on a GPU cluster. While such rigorous training is useful to update inference parameters of the CNN via OTA protocols on a regular basis (weekly or monthly), our work aims to provide instant on-vehicle training to better handle a similar critical situation if encountered again on the *same day, even within minutes*.

To account for limited on-board compute and storage , a pre-trained CNN could be exposed to just a few new images, but such attempts have been shown to result in "catastrophic forgetting" — the prediction accuracy of already learned features drops significantly — known as the stability-plasticity dilemma [13].

In this paper, we propose CLAIRE, a continual learning real-time framework based on YOLO [11], which enables the on-board system to learn new scenarios from a few image frames in a very short time. The novel lightweight architecture is designed to elastically support continual learning from new data. In addition, this architecture design can be applied to any general one-stage CNN-based object detector. We demonstrate that our design fits on-board continual learning for autonomous driving and still meets real-time constraints in a sensitivity study. We compare our work with existing continual learning methods in object detection, i.e., joint-training and state-of-the-art feature-reweighting [14], in both timing and accuracy. Experimental results show that our method outperforms others in both learning time and detection accuracy and only incurs a 3.25ms delay in real-time deep learning inference.

The rest of the paper is organized as follows. Related work on continual learning algorithms and real-time objection detection is discussed in Section II. Section III introduces our novel continual learning framework, CLAIRE, detailing both design and implementation. In Section IV, the timing and accuracy of surveyed methods and our newly proposed method are compared. Also, a sensitivity study is performed on our proposed method. Section V summarizes the contributions and discusses the potential directions of future work.

II. Related Work

Object Detectors: As addressed in Section I, deep CNN based objectors can be divided into two-stage and one-stage detectors. Two-stage detectors [8] and their variants [9], [10], [15]–[19] identify regions of interest (containing some object) in the first stage, and send the region proposals to the second stage for object classification and localization. Single-stage detectors can be anchor-based (YOLO [6], [7], [11], [20], SSD [12], RetinaNet [21], RefineDet [22]) or point-based (CenterNet [23], CornerNet [24], FCOS [25]) and are capable of predicting the categories and locations of objects without the region proposal stage. The majority of research has focused on optimizing one-stage detectors, in both accuracy and speed, to make them a better fit for ADS. Nowadays, Tesla Autopilot [4] and Baidu Apollo [26] leverage the one-stage detector due to its higher performance and accuracy compared to a two-stage detector.

Continual Learning: The challenge of continual learning with deep CNN models is catastrophic forgetting [13]. We investigate the forgetting problem caused by direct continual learning, a crucial and general problem in incremental learning with CNNs. Well-known image classification ML methods focus mainly on learning new categories while preserving previously learned knowledge with different techniques, e.g., via ensemble modeling [27]–[29], transfer learning [28], [30], fine tuning [31]–[35], distillation or data exemplars from previously learned knowledge [30], [31], [36], [37], attention-based meta-learning [38], [39], and addition or adaption of the network architecture [40]. Bi-objective [41] focuses on learning new instances instead of new classes.

Object detection models leverage deep CNN as well. RILOD [29] follows the incremental learning methodology from [31] and applies to one-stage anchor-based object detection. Kang at al. [14] propose a module to change weights in the YOLO-based architecture to recognize novel categories from few images. ONCE [42] and PNPDet [43] leverage the meta-learning and dedicated sub-network techniques, respectively, for incremental learning in a point-based one-stage detector. Our work focuses on the object detection's capability to learn additional data to improve the accuracy of *existing categories*. Both of these learning techniques suffer from potential memory loss triggered by the change in distribution of in continual learning, which is the root cause of forgetting.

Lightweighed Models: PatDNN [44], SqueezeNet [45], HashedNets [46], Xnor-Net [47], MCDNN [48] and Deep Compression [49] take advantage of pruning, compression, asymmetric encoding, and related techniques to minimize the size of ANNs such that a lighter weight ANN can be deployed on smartphones, FPGAs, and other embedded devices. SubFlow [50] dynamically selects sub-graphs of ANNs for both training and inference based on real-time constraints so that training/inference can meet deadlines. However, all

these works suffer from a speed-accuracy tradeoff and do not address continual learning.

Multi-branch Inference: HydraNets [51] proposes a dynamic network architecture template for efficient inference of image classification. This architecture contains multiple branches specialized for disjoint categories, where a gate chooses which branches to run when performing inference. However, this work assumes the inference branches have already been learned conventionally offline. In contrast, we continually train with new images of *existing* classes and employ a novel dual-head architecture instead, which aims to reduce training cost but also to keep inference cost low.

III. Design and Implementation

Our work aims to create the novel ability for autonomous vehicles to learn new driving scenes on-the-fly in minutes by updating a model incrementally on-board rather than waiting weeks for any periodically retrained model from the cloud. Current object detectors fail to support such a capability. This section describes the design and the implementation of our novel CLAIRE framework.

A. Dual-head Architecture Design

Yolov4 [11] optimizes both accuracy and speed of object detection. Modern one-stage detectors usually consist of two parts, a backbone that automatically extracts features from input images and a detection head that predicts the category and location of objects within an image. Yolov4 leverages the same network architecture as Yolov3 [7]. In addition, Yolov4 employs deeper layers in the backbone than that of Yolov3 by optimizing the activation function and the bounding box selection algorithm.

We designed a novel dual-head object detection architecture that enables real-time incremental learning with new images of existing classes based on the Yolo network architecture.

Our CNN backbone (see Figure 1) extracts features from an input image classified as low-level, medium-level, and high-level features output (corresponding to O1, O2, and O3, respectively, in the figure). These feature outputs are the inputs of our multi-scale detection comprised of the original base head for all categories (left part in blue color bounding box) and the incremental head specializing in learning new scenario data (lower right part in green color). When an OTA update is received, the base head is reset to the newly received base model if and only if this new model includes the incrementally learned scenarios, i.e., if the last cloud training cycle included the uploaded images that were subject to prior on-board training of the incremental head.

Every detection head (both base and incremental) consists of three prediction blocks dedicated to detecting an object of small (S), medium (M), and large (L) size, respectively. Each prediction block includes eight cascaded convolutional layers and a final detection layer. The block that predicts larger-sized objects depends on the input not only from the backbone output but also on the output from the smaller (predecessor) prediction block, i.e., the input of M depends on the output of S, and L depends on M (horizontal arrows in Figure 1 between two blocks).

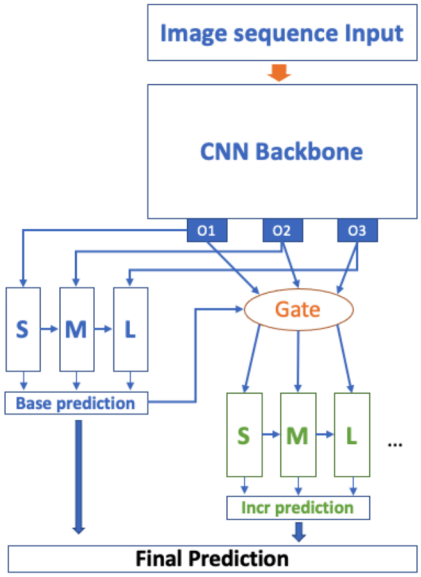

Fig. 1. CLAIRE framework

Our design differs from HydraNet [51] that consists of multiple prediction branches (or heads). Each HydraNet branch is dedicated to predicting similar categories. In the early stage of our work, we experimented with a multi-head architecture, where each detection head was to detect a subset of the classes. However, this design was deemed infeasible for on-board real-time autonomous driving since the detection time is proportional to the number of heads. Considering the limited on-board GPU and memory resources, more heads imply a need for more memory to store weights and for more compute resources. This led to much higher inference/testing time without increasing accuracy or prediction compared to our dual-head design.

B. Continual On-board Learning

The objective of the base head is to detect all n classes. The base head model is an on-vehicle detector that is periodically updated from rigorous re-training in a data center. Both the base and incremental heads are sharing the same backbone in prediction, so that training the base head and the backbone of the CNN are a prerequisite for training incremental heads and is performed offline. Back propagation is applied on all convolutional layers when training the base head. Thus, the base model consists of the backbone and the base detection head.

The incremental head is online trained with few images per class at a time, which reflects resource scarceness (in storage and computation) of on-board devices. The incremental head is trained for all current categories. During the continual learning process, the architecture initializes a new specialized incremental head for training. Backward propagation and update operations are not performed on the backbone during continual

learning. Instead, only the weights of convolutional layers in the detection head (green boxes in the figure) are updated. This has two advantages: First, accuracy is guaranteed because we share low level features and update high level ones during continual learning of new data to fine-tune the incremental head. Second, without any back-propagation and backbone updates during training, computation power is conserved while also reducing training time significantly. Since the volume of data for incremental training is small, we leverage data augmentation (i.e., image cropping, flipping, rotating, etc.) to increase the diversity of input images. This data augmentation is also applied to the training of the base head.

Any periodic OTA update can still be applied to the base model in the proposed architecture since we do not modify the structure of the backbone and the base detection head, i.e., the base model network architecture is retained. Thus, weights from both offline retraining with massive old/new data and online/on-board training with (smaller) new data are accommodated simultaneously for the dual-head design.

C. Online Testing:

During inference/testing, image sequences are sent to the backbone of the network. The backbone is responsible for extracting features per image. During this process, the low-, medium-, and high-level feature outputs are routed to the associated detection blocks of the base head while the backbone caches the intermediate output state. Both the base and the incremental heads are activated during the online inference. The purpose of the gate function is to choose the top detection from the final layers with the highest probability. For example, detecting a single object may have multiple prediction results in both class IDs and locations. The gate selects the best among all of these results.

The output of the base head is a list. Each item within the list includes objectiveness score, class scores, and coordination information of predicted bounding box. The algorithm first filters out detection with predicted objectiveness score lower than a pre-defined threshold value, and then uses quicksort to order the detections by descending class score. In this way, the detection with highest class score is first in the list. Any subsequent list item is compared to the class score of the first item, and if they differ by only a small threshold ϵ, the corresponding (top-k) classes are activated for detection. If the difference exceeds ϵ, the loop is terminated and the selected classes are returned (as differences only increase in subsequent iterations). The gate selection is presented in Algorithm 1.

The final prediction bar at the bottom of Figure 1 combines the outputs from both base and incremental heads and selects the prediction output with maximum probability, i.e., either a class from the base or from the incremental head. This fusion step guarantees that prediction outputs enjoy the best of both worlds. After all, the incremental head may not always predict the best result for all data.

Algorithm 1 Select Class IDs

```
for i = 0 to N − 1 do
  if detections[i].objscore < threshold then
    detections.remove(i)
  end if
end for
Sort detections in descending order by class score
for i = 1 to N − 1 do
  if (detections[0].cs − detections[i].cs) < ϵ then
    selectedCls.insert(detections[i].clsId)
  else
    break
  end if
  return selectedCls
end for
```

D. Implementation

The implementation of our method is based on the state-of-the-art real-time object detector framework Yolo [7]. It is compatible with Yolov3 and Yolov4, since both use the same architecture in the detection head. The framework is written in C and CUDA, as is our implementation of the incremental multi-scale head. The incremental learning based on data from a single or more classes is realized by appending an incremental detection head to the end of the current backbone network. Three-level feature outputs are consolidated within each head to provide a single prediction. The incremental head can be dedicated to a subset of the n classes, which differs from the base head that covers all n classes. A reflection interface implemented via a member variable, the so-called classid, identifies the classes of a given head (or returns the base as an identification). This facilitates (a) the selection of classes using the output of the base head and (b) the identification of the prediction class resulting from multiplexing of final predictions from base and incremental heads.

The gate activates any incremental detection classes based on the objectiveness scores and class probability from the prediction of the base head. If there is an "approximate tie" in the selection condition, the gate triggers multiple corresponding incremental classes simultaneously. An approximate tie exists if the probabilities of the top class and another class differ by less than some threshold ϵ. The implementation uses an ϵ of 10%. Additionally, if the base head fails to detect a class altogether, all classes within the incremental head are activated. The gate function can also selectively only trigger the base head if and only if this newly OTA updated model includes the incrementally learned scenarios. This avoids duplicate predication on the incremental head, thereby reducing the inference cost.

Since the incremental head has the same number of layers, any selected classes are mapped to a range of layers within the network. This allows an incremental head to automatically obtain (a) input from layers belonging to the backbone and (b) forward output of the incremental head to the final prediction multiplexer. These capabilities are replicated in C and CUDA

for both CPU and GPU computation. A local barrier within each head consolidates the output from the prediction layers corresponding to the small/medium/large image sizes while a global barrier within the final prediction block ensures that the multiplexer provides as a final output the highest prediction probability among both heads (base and incremental ones).

Testbed Experiments were conducted on an X86_64 platform with two Intel Sandy bridge processors (with a combined 16 cores) utilizing 16GB DDR3 1600 ECC DRAM and a Nvidia RTX 2070 GPU with 8 GB of memory. The CUDA and CUDNN version are 10.0 and 7.4, respectively, running on a CentOS 7.7.1908 distribution with a 4.10 Linux kernel. This setup provides GPU acceleration for both training and inference/testing, both of which exploit the linear algebra libraries provided by Nvidia. Our source code and dataset are available on the Github repository.

IV. Experiments

A. Dataset

The dataset for both training and testing is Microsoft's Common Objects in Context (COCO) [52]. As indicated by its name, images in the COCO dataset are taken from everyday scenes augmented by a "context" to the objects captured. There are 80 object categories in COCO. We extract the object label from the annotation file associated with the image and customize the label format based on our needs. Since our research focuses on autonomous driving, we use a subset of COCO categories related to driving. Other categories, such as food, appliance, and kitchenware, do not contribute to autonomous driving. A reduction in categories also significantly reduces the training time and, to a smaller extent, inference cost as the total number of images is reduced. This facilitates experimentation with a larger set of model variations.

We leverage COCO's APIs to filter out excluded categories as we select the relevant subset of COCO. To meet the image label requirement of the Yolo model, we convert the label from COCO to Yolo format, i.e., each image is associated with a corresponding label file (distinguished by file extension). A label file has one or more entries, each representing a five tuple to identify an object with a class ID, object center (x,y coordinates), width, and height.

Table I depicts the number of images (for training and testing/inference) and the number of objects in them, where images may contain multiple objects that can partially overlap, i.e., only parts of an object are visible.

TABLE I
Dataset details

Class ID	# train imgs	# train objs	# test imgs	# test objs
0	2287	4955	1098	2439
1	8680	19741	6117	19230
2	2442	6021	1200	3023
3	2243	3833	840	1445
4	2791	4327	1044	1935
5	2464	3459	1278	1599
6	4321	7050	1602	2923
7	2098	7590	1046	3597
8	2893	9159	1116	3472
9	1205	1316	570	628
10	1349	1372	454	594
11	481	833	257	501
12	3844	6571	1958	3494

B. Evaluation Metrics

We use the conventional ML metrics of precision, recall, and F1-score, detailed in the following, to assess experimental results.

The Intersection over Union (IoU) measures the overlap between the predicted bounding box and the ground truth box divided by the union of the two boxes in object detection. We include IoU as a metric since the predicted location of an object is of significant importance in autonomous driving, and it affects object classification. In the experiments, we use a uniform IoU threshold 0.5 for all incremental learning methods. If the IoU value exceeds the threshold, an object is detected; otherwise, no object is detected.

A model recognizes an object detection as a true positive (TP) when the object is detected (IoU greater than the threshold) and classified correctly. For detecting class A, the predicted result counts as a false positive (FP) when the object is recognized as A but the ground truth is $\neg$ A. There are two cases counting as a false negative (FN). One is no detection and the other is detected as $\neg$ A while the ground truth of the object is A for both cases.

We also measure the training and inference time for each continual learning method. Training time is important since we want to start using the new model with additional capabilities as soon as possible. While continual learning is not subject to the hard real-time constraints (in contrast to inference), any training performed at a lower priority in the background as best effort should still provide a new model within minutes. Inference can then switch over to the new model, and automated driving decisions may benefit from earlier object detection in high-level decision making. Inference time is crucial as well since it has hard real-time constraints, where the continually learned on-board model has to predict the result by a deadline (before the next image frame arrives).

C. Analysis of Our Dual-Head Method

We compare our dual-head architecture with the state-of-the-art work called feature reweighting [14] due to the following considerations: First, based on our literature survey, all continual learning approaches on object detection focus on learning new classes. Yet, our method is on continually learning new data with different distributions for existing classes. Second, although it adapts the model to new classes, feature reweighting is the most relevant work to compare to. It only uses a small amount of new data like ours to train the model continually. This is practical in an onboard continual learning scenario. Their approach is based on the same object detection model as ours. Third, feature reweighting has released their source code, which allows us to reproduce

their approach. We have also compared our method with joint training. Our approach has almost the same accuracy with that of joint training. Due to space constraints, we have to omit the experimental result of joint training in the following figures.

We next assess detection accuracy with newly learned data, never seen before, and seen before data. A comparison in terms of timing analysis is given Sec. IV-D (Timing Analysis). In this comparison, the base method is trained with the full base data as it is not subject to continual learning. Instead, it should be considered an OTA update resulting from vendor-initiated periodic retraining in their data center, with both old and new data from the field collected over longer periods of time (weeks to months). We take a snapshot of such a base model as a baseline to compare to the two continual learning methods, (1) feature reweighting and (2) our dual-head method. These two are trained with the same baseline data plus new data under continual learning. The hyperparameters are unchanged and remain the same for all methods.

Evaluation with New Data: The purpose of this experiment is to determine how well the dual heads have learned the new data during continual training. We use 20 new images from each class to train the incremental detection heads and thus assess our dual-head architecture. As shown in Figure 2, the x-axis denotes the class ID. Compared to the base detection head (top) that has never seen these data, our dual-head method (bottom) has detected 0%–150% more TPs and dramatically reduced the number of FNs. Moreover, there is a decrease in the number of FPs 0% – 22%. The precision of the dual head has increased by 1%–13% across all n classes, and the recall of the dual head has improved by 17% – 59% for all n classes.

Although the recall of the reweighting method (middle) remains about the same as that of the base model across all n classes, accuracy drops significantly among all classes due to the high number of FPs, which are misclassifications. For detection of a ground truth with only one class (id = 1), the reweighting method tends to predict multiple bounding boxes with classes (nearly all n), thereby generating n times more FPs for every class. Our method performs detection, on average, 61.4% better in terms of precision and 44.4% in recall over all classes than that of the reweighting method. The results show that the incremental head has learned new data for all n classes. This is the key advantage of the dual-head method over other methods, without access to previously trained data.

Evaluation with Never Seen Before Data: Figure 3 depicts the detection metrics of n classes contrasting the base, reweighting, and dual-head methods for the given testing data (not subject to prior training).

Our dual-head detection architecture results in a larger number of TP cases and fewer FNs across all n incrementally learned classes. Although our method incurs a slightly higher FP rate than that of base-head detection, the average precision over all n classes is 3.4% lower, with a standard deviation of 0.03. The dual head resulted in 247 more TPs than the base head. The average recall over all n classes is 1.1% higher than that of base-head detection, with a standard deviation of 0.01. However, the average precision and recall over all classes of the reweighting method is 51.8% and 14.5% lower than those of the base model. The prediction results of the reweighting model have both high FP and FN, which is an indication forgetting issues during continual learning. This is rooted in their method that focuses on learning new data from new classes instead of learning new data from existing classes. If there is not new class to learn, the reweighting factor balances across all n classes, which makes the model associate an object with all n classes, leading to high FP for all classes. In contrast, our continual dual-head method has learned new data with high precision and recall while still retaining prior levels for both metrics of the base method for other test data. Moreover, its testing results on new data outperform that of the feature-reweighting method in Figures 2 and 3.

Sensitivity Analysis: The purpose of this experiment is to test the robustness of our dual-head method, i.e., to ensure that continual learning over many images does not result in catastrophic forgetting of learned features. This sensitivity study exposes the base to all training data. The incremental head is then continually trained with 20 new images of each class. We then expose this trained dual-head system in testing (inference) to the same training data across all n classes. Notice that this is only done as a validation experiment to assess the ability to retain prior seen knowledge. Figure 4 (bottom) depicts detection results of our dual-head architecture over all n classes indicated by their index (see dataset details in Table I).

Results of the dual-head configuration remain nearly the same across classes as for the base head, i.e., continual training retains prior knowledge across original and retrained classes. The average precision and recall difference is 4.3% and 0.1%, respectively, with a standard deviation of 0.03 and 0.003. In contrast, the average precision and recall difference is 59.9% and 26.5% between the base model (top) and the reweighting method (middle). The experiment in Section IV-C (New Data) additionally showed that new knowledge could also be retained. In other words, our dual-head method is not subject to memory loss with respect to already learned features.

D. Timing Analysis

In this section, we compare both learning and testing/inference time among multiple learning methods and assess why our continual learning method with its dual-head architecture is the fastest. We also conduct a preliminary experiment that executes both online testing/inference and continual learning instances on the same machine to outline directions of future work.

Training Time: The second column of Table II shows the single image re-training time of each continual learning method performed on our testbed. The indicated training time is per image. This implies that the model retraining time for the base method is much longer since it is exposed to both new *and* old data.

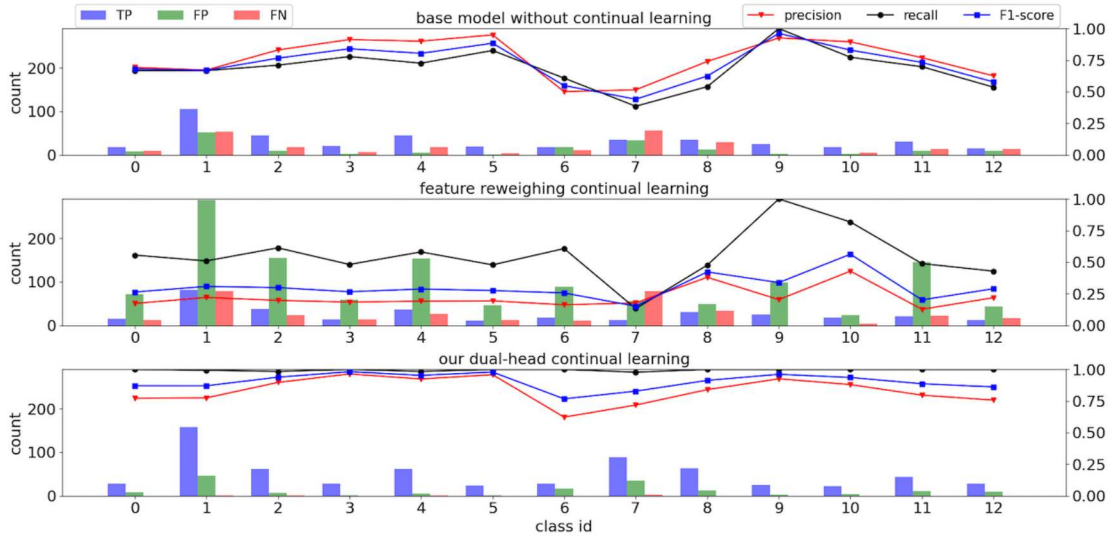

Fig. 2. Evaluation with new data

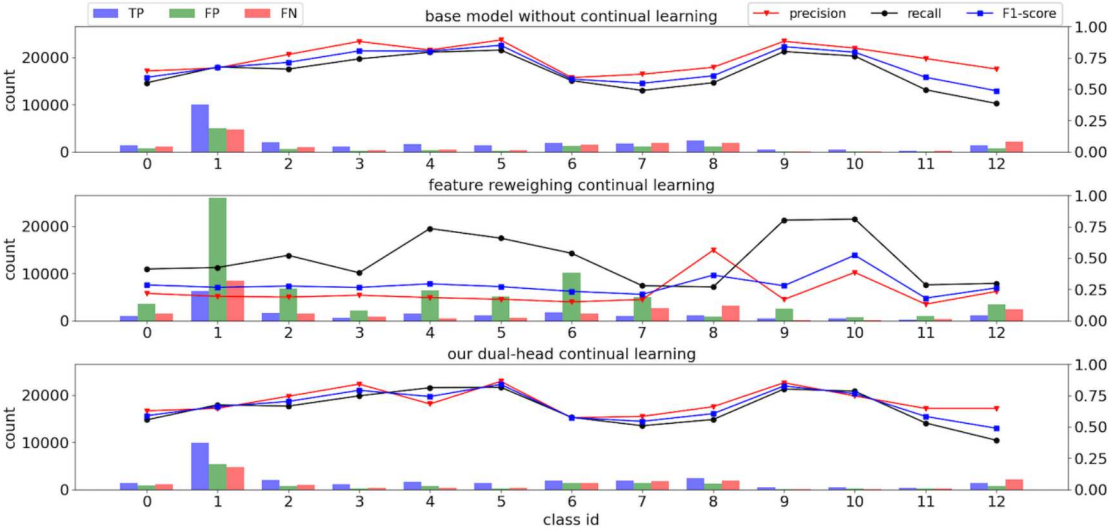

Fig. 3. Evaluation with never seen before data

Re-training cost is a function of both data volume and computation. While the computational overhead of direct learning and joint training per image through the CNN layers is the same, a different number of images is used in re-training. Direct continual learning only uses images of new scenarios while joint training exposes the same fraction of images across *all* classes, a 13-fold increase in data. The runtime costs in the figure indicate that this data volume causes joint training to take 16 times longer than direct continual learning. The discrepancy (13X data vs. 16X runtime) is due to less I/O caching in the operating system layer but also increasingly denser matrix operations inside the CNN of the latter — resulting from data not being evenly distributed across categories, as we uncovered. Changing weights of features [14] leverages an additional CNN-based module to assist in incremental learning. Although it can learn new classes from a small amount of images, the model still needs to access old data in both learning phases. This makes the training time of the reweighting method longer (by 3X).

The dual-head method has the lowest re-training cost, only 50% that of direct learning. It follows the same computation pattern and data volume as Direct, but without back propagation on the backbone. In addition, the number of filters in the last convolutional layers is reduced since a head may only learn a subset of classes. Learning new images in the order of seconds can be feasibly done as a background task, or even when a vehicle is not in motion, without straining its computational and energy resources.

Finally, learning from scratch by training with all images under the YOLO base framework takes 33hrs, 50min (not

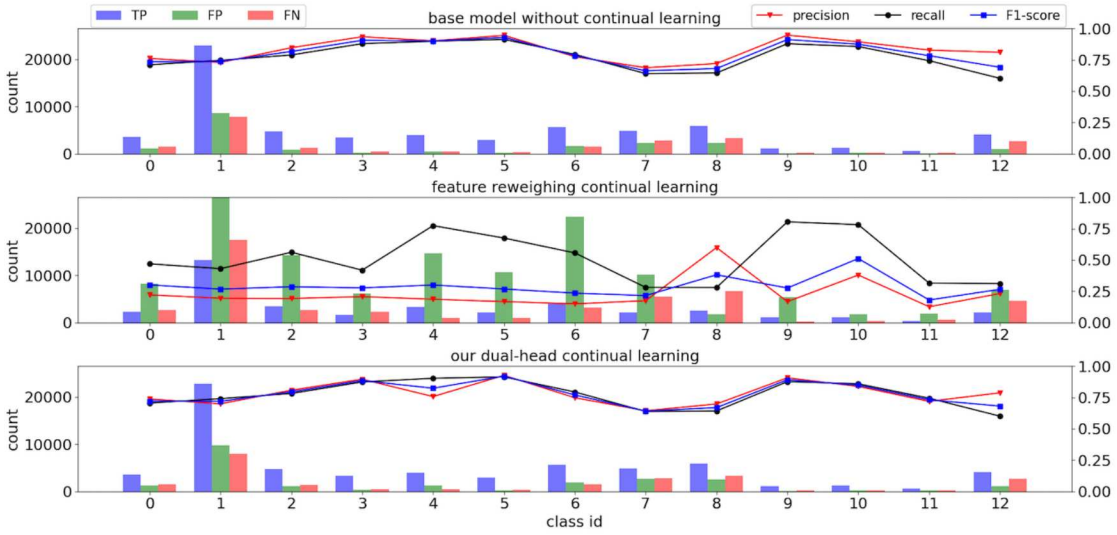

Fig. 4. Evaluation with seen before data

shown in the table), requires excessive storage and consumes considerable energy. This underlines the advantages of incremental learning with our novel dual-head method.

TABLE II
TIME DETAILS

Method	Training (s)	Detection (ms)	Speed (FPS)
Base	4.61	18.76	53.29
Direct	4.61	18.15	55.09
Joint	73.80	18.67	53.54
Reweight	13.80	581.8	1.95
Dual-head	**2.34**	22.25	44.94

Testing/Inference Time: As shown in third column of Table II, the average inference time of conventional continual learning methods (direct and joint) is under 19ms per image. The test dataset has a total of 11,404 images containing 41,495 objects. While base, direct and joint methods meet real-time constraints at over 50fps, their precision and accuracy are unacceptable compared to our dual-head design. While feature reweighting recognizes new objects, it fails to meet real-time requirement falling short of 2 FPS. Its main computation overhead originates from generating weight vectors from all trained data. This slow speed and excessive access to all old data render the method unsuitable for on-board continual learning. Our proposed method has a 3.5ms higher detection time than the average time of conventional methods, but still within real-time constraints at 45fps with at least equally high precision and accuracy than any other method. Its slower frame rate is due to having to forward through more layers than the base method.

V. CONCLUSION

This work contributes (1) the design and implementation of a real-time on-board continual framework that learns within seconds, (2) a novel dual-head architecture to support real-time object detection with new images of existing classes, (3) a fair comparison in timing and accuracy between these methods in the context of continual learning for autonomous vehicles, and (4) a sensitivity study on the quality and quantity of continually learned images for our dual-head technique, including its real-time suitability.

Our method overcomes the memory loss problem of knowledge from old data and performs well in both training and inference. These properties plus its fast learning capability make the proposed method the best candidate for deployment in autonomous driving, where continual learning is performed in the background within a short time while retaining realistic frame rates for real-time foreground object detection.

ACKNOWLEDGMENT

This research was supported in part by NSF grant 1813004.

REFERENCES

[1] "Intel mobileye," http://www.mobileye.com, 1999.
[2] "Nvidia drive os," https://developer.nvidia.com/drive/driveos, 2019.
[3] "Volkswage vw.os," https://www.volkswagenag.com/en/news/fleet-customer/2021/01/transformers.html, 2021.
[4] "Tesla autopilot," https://www.tesla.com/autopilot, 2015.
[5] "Android auto os," https://developers.google.com/cars/design/automotive-os, 2016.
[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[7] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[8] 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.
[9] R. Girshick, "Fast r-cnn," in *IEEE international conference on computer vision*, 2015, pp. 1440–1448.
[10] 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 and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[13] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects," *Frontiers in psychology*, vol. 4, p. 504, 2013.

[14] B. Kang, Z. Liu, X. Wang, F. Yu, J. Feng, and T. Darrell, "Few-shot object detection via feature reweighting," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[15] J. Gao, J. Wang, S. Dai, L.-J. Li, and R. Nevatia, "Note-rcnn: Noise tolerant ensemble rcnn for semi-supervised object detection," in *Proceedings of the IEEE international conference on computer vision*, 2019, pp. 9508–9517.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[17] Z. He and L. Zhang, "Multi-adversarial faster-rcnn for unrestricted object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6668–6677.

[18] T. Wang, J. Huang, H. Zhang, and Q. Sun, "Visual commonsense r-cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 760–10 770.

[19] T. Wang, X. Zhang, L. Yuan, and J. Feng, "Few-shot adaptive faster r-cnn," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7173–7182.

[20] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[22] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4203–4212.

[23] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[24] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[25] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[26] "Apollo, an open autonomous driving platform," http://apollo.auto/, 2019.

[27] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[28] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *IEEE International Conference on Computer Vision*, 2017, pp. 3400–3409.

[29] D. Li, S. Tasci, S. Ghosh, J. Zhu, J. Zhang, and L. Heck, "Rilod: near real-time incremental learning for object detection at the edge," in *ACM/IEEE Symposium on Edge Computing*, 2019, pp. 113–126.

[30] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[31] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[32] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *National academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[33] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7765–7773.

[34] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 67–82.

[35] A. Rosenfeld and J. K. Tsotsos, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[36] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *European Conference on Computer Vision*, 2018, pp. 233–248.

[37] A. Chrysakis and M.-F. Moens, "Online continual learning from imbalanced data," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. Virtual: PMLR, 13–18 Jul 2020, pp. 1952–1961. [Online]. Available: http://proceedings.mlr.press/v119/chrysakis20a.html

[38] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4367–4375.

[39] M. Ren, R. Liao, E. Fetaya, and R. Zemel, "Incremental few-shot learning with attention attractor networks," *Advances in Neural Information Processing Systems*, vol. 32, pp. 5275–5285, 2019.

[40] S. W. Yoon, D.-Y. Kim, J. Seo, and J. Moon, "XtarNet: Learning to extract task-adaptive representation for incremental few-shot learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. Virtual: PMLR, 13–18 Jul 2020, pp. 10 852–10 860. [Online]. Available: http://proceedings.mlr.press/v119/yoon20b.html

[41] X. Tao, X. Hong, X. Chang, and Y. Gong, "Bi-objective continual learning: Learning 'new'while consolidating 'known'," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5989–5996.

[42] J.-M. Perez-Rua, X. Zhu, T. M. Hospedales, and T. Xiang, "Incremental few-shot object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 846–13 855.

[43] G. Zhang, K. Cui, R. Wu, S. Lu, and Y. Tian, "Pnpdet: Efficient few-shot detection without forgetting via plug-and-play sub-networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 3823–3832.

[44] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," *arXiv preprint arXiv:2001.00138*, 2020.

[45] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[46] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International conference on machine learning*, 2015, pp. 2285–2294.

[47] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[48] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *International Conference on Mobile Systems, Applications, and Services*, 2016, pp. 123–136.

[49] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[50] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2020.

[51] R. T. Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[52] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.