RTScale: Sensitivity-Aware Adaptive Image Scaling for Real-Time Object Detection

Author(s):
Heo, Seonyeong; Jeong, Shinnung; Kim, Hanjun

Publication date:
2022

Permanent link:
https://doi.org/10.3929/ethz-b-000558309

Originally published in:
Leibniz International Proceedings in Informatics (LIPIcs) 231, https://doi.org/10.4230/LIPIcs.ECRTS.2022.2

RTScale: Sensitivity-Aware Adaptive Image Scaling for Real-Time Object Detection

Seonyeong Heo
Department of Information Technology and Electrical Engineering, ETH Zürich, Switzerland

Shinnung Jeong
Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea

Hanjun Kim
Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea

Abstract

Real-time object detection is crucial in autonomous driving. To avoid catastrophic accidents, an autonomous car should detect objects with multiple cameras and make decisions within a certain time limit. Object detection systems can meet the real-time constraint by dynamically downsampling input images to proper scales according to their time budget. However, simply applying the same scale to all the images from multiple cameras can cause unnecessary accuracy loss because downsampling can incur a significant accuracy loss for some images.

To reduce the accuracy loss while meeting the real-time constraint, this work proposes RTScale, a new adaptive real-time image scaling scheme that applies different scales to different images reflecting their sensitivities to the scaling and time budget. RTScale infers the sensitivities of multiple images from multiple cameras and determines an appropriate image scale for each image considering the real-time constraint. This work evaluates object detection accuracy and latency with RTScale for two driving datasets. The evaluation results show that RTScale can meet real-time constraints with minimal accuracy loss.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Parallel architectures; Software and its engineering → Real-time systems software; Computing methodologies → Neural networks; Computing methodologies → Object detection; Theory of computation → Scheduling algorithms

Keywords and phrases Real-time object detection, Dynamic neural network execution, Adaptive image scaling, Autonomous driving, Self-driving cars

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.2

Supplementary Material *Software (Source Code)*: `https://github.com/seonyheo/darknet`

Funding This work is supported by IITP-2020-0-01847, IITP-2020-0-01361, and IITP-2021-0-00853 through the Institute of Information and Communication Technology Planning and Evaluation (IITP) funded by the Ministry of Science and ICT. This work is also supported by Samsung Electronics.

Acknowledgements We thank the anonymous reviewers for their valuable feedback. We also thank the CoreLab members for their support and feedback during this work. (Corresponding author: Hanjun Kim)

1 Introduction

Real-time object detection in autonomous driving is crucial to avoid severe accidents. Autonomous cars have multiple cameras around their bodies [39] and use the cameras to detect objects on the road. Based on the object detection results, autonomous cars make decisions on how to control their braking system and steer their wheel. Since object detection provides essential visual information for autonomous cars, object detection must finish on time to make timely decisions. If autonomous cars fail to make decisions on time, they may hit pedestrians or other cars. Therefore, autonomous cars should detect objects on the road timely and accurately.

Recent advances in deep neural networks (DNNs) have brought real-time object detection into reality. With tens or hundreds of neural network layers, object detection networks predict regions on the input image (i.e., bounding boxes) that are likely to contain an object, and classify their object categories. In general, we call a whole end-to-end object detection network, including bounding box prediction and classification as an *object detector*.

Prior work [14, 13, 34, 17, 32, 36, 35, 6, 44] has proposed DNN architectures for object detection. Especially, one-stage object detectors such as SSD [32] and YOLO [34] enable fast and accurate object detection by integrating bounding box prediction and classification. For example, YOLO (You Only Look Once) achieves up to 45 frames per second with high accuracy [34]. Therefore, open-source autonomous driving platforms such as Autoware [24, 25] and Apollo [1] use YOLO-based networks for object detection.

Aside from designing a novel DNN architecture, various optimization techniques are available to enhance the detection speed of existing object detectors. One of the most popular approaches is model compression [16, 15, 28, 10], which reduces the computational cost by pruning parameters or using lower-precision numerics. However, the model compression techniques require fine-tuning the target network to minimize accuracy loss, so they could hardly reflect time-varying real-time constraints. Another approach is dynamic image scaling [8, 7], which can reduce the computational cost by dynamically downsampling input images. By adjusting the image size according to the time budget, dynamic image scaling can help satisfy dynamically changing real-time constraints.

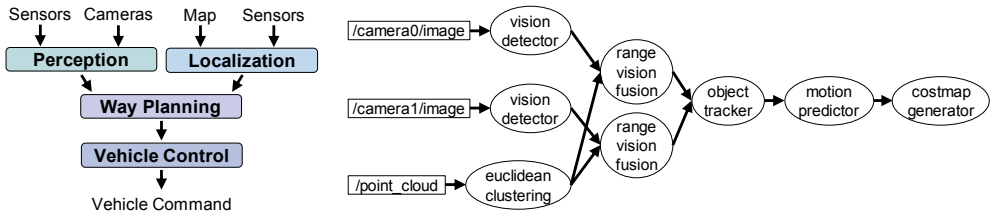
However, simply applying dynamic image scaling to object detection systems can cause unnecessary accuracy loss. With the state-of-the-art object detectors [6], we observe that each image has a different sensitivity to image scaling. Whereas image downsampling causes significant accuracy loss for some images, it barely causes accuracy loss for the other images. If an image only contains objects that are easy to detect even at a low scale, the image tends to be less sensitive to downsampling. Therefore, to reduce accuracy loss in image downsampling, it is necessary to consider the sensitivity in determining an appropriate scale for each image.

This work proposes RTScale, a new sensitivity-aware adaptive image scaling scheme for real-time object detection, which applies different scales to different images reflecting their sensitivities to image scaling and time budget. RTScale extends an existing object detector with a new scale sensitivity inference module and minimizes its overhead by reusing image features from the object detector. While offline, RTScale trains the sensitivity inference module to dynamically infer the impact of image scaling on the accuracy. While online, RTScale infers the scale sensitivities for multiple images from multiple cameras with the trained module, and determines appropriate image scales for the images considering the real-time constraint.

This work evaluates object detection accuracy and latency with two driving datasets: KITTI MOT [12] and BDD100K MOT [43]. This work implements RTScale on top of the state-of-the-art object detection framework [11]. The evaluation results show that RTScale can infer the sensitivity of images with low overhead and meet real-time constraints with minimal accuracy loss compared with another adaptive scaling scheme.

The contributions of this work are:

- Sensitivity-aware adaptive image scaling scheme for real-time object detection
- Sensitivity inference module which infers the scale sensitivity of an image based on its features
- Evaluation of the proposed approach with two real-world driving datasets



(a) Full pipeline. **(b)** Perception pipeline.

Figure 1 Autonomous driving and perception pipeline.

2 Background and Motivation

2.1 Autonomous Driving and Object Detection

In general, autonomous driving consists of four primary jobs: perception, localization, way planning, and vehicle control. Autonomous driving systems first perceive their surrounding environment with cameras and sensors. At the same time, autonomous driving systems localize their locations using high-definition maps and sensors. Based on the perception and localization results, autonomous driving systems plan how to reach the destination avoiding obstacles on the road. Finally, autonomous driving systems determine how to control the vehicle according to the plan. Figure 1a briefly illustrates the autonomous driving pipeline.

Among the four primary jobs, perception plays an essential role in providing safe autonomous driving. Perception offers visual information on the road, such as whether pedestrians are near and how many cars are on the road. In practice, since high-definition maps only provide static information of the road, such as the locations of traffic lights, autonomous cars cannot solely rely on high-definition maps for safe autonomous driving. Through the perception process, autonomous cars can detect dynamic obstacles on the road and make the right decisions on their next movements. If autonomous cars cannot perceive the surrounding environment in real time, autonomous cars would fail to avoid obstacles on the road.

For perception, autonomous driving systems should process multiple camera images in practice. According to Tesla Model S owner's manual [39], Tesla Model S cars use eight cameras for autonomous driving: one camera above the rear license plate, two cameras on each door pillar, two cameras on each front fender, and three cameras on the wind shield. According to prior work [29, 9], autonomous driving systems should finish the end-to-end processing within 100 ms. Therefore, autonomous driving systems should process all the images from multiple cameras within less than 100 ms.

Typically, autonomous driving systems implement perception with various computer vision algorithms such as object detection and object tracking. Figure 1b summarizes the perception pipeline process of Autoware [24, 25], one of the most popular open-source autonomous driving systems. When the system receives the images and point clouds from cameras and LIDAR, the system detects objects with the images and the point clouds. Then, the system fuses the objects from the images with the objects from the point clouds. Next, the system applies object tracking and motion prediction algorithms to the fused objects. Finally, the system obtains a cost map for way planning, which indicates the drivable area around the autonomous car.

In the perception process, object detection becomes the performance bottleneck incurring the most computational overhead. Figure 2 shows the profiling result of each task in the perception pipeline. For profiling, this work instruments time measuring code into the

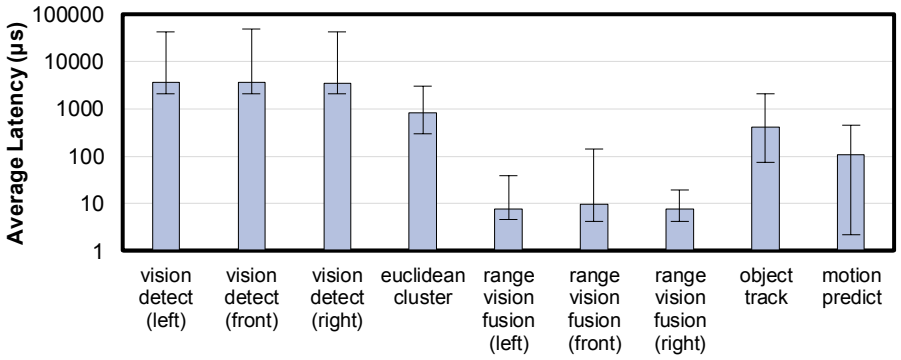


Figure 2 Latency profile of tasks in the perception pipeline in Figure 1b. (416, 416) image scale is used in object detection. Note that the y-axis is logarithmic.

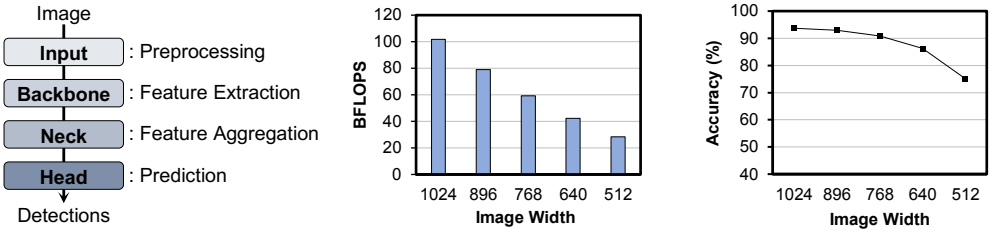
Autoware perception module [2]. Note that this work only measures the latency of inference, excluding queuing delay and communication time. In the graph, each error bar indicates the maximum and minimum latency of each task. The graph shows that (vision) object detection takes 4 ms while the other tasks take less than 1 ms on average. Therefore, the profiling result demonstrates that object detection takes the longest time among the different tasks in the perception pipeline.

2.2 Existing Object Detection Networks

In recent years, deep neural networks (DNNs) have remarkably enhanced both speed and accuracy of object detection. Figure 3a briefly describes the general structure of object detectors. An ordinary object detector comprises four parts: input, backbone, neck, and head [6]. The input part takes an image as an input and preprocesses the image. The backbone part contains a deep convolutional network such as ResNet [18] to extract the features of the input image. The neck part typically contains a small network that collects the features from different backbone stages such as FPN [30]. Finally, the head part predicts the location and category of objects.

There are two primary types of object detection networks: two-stage networks (e.g., R-CNN series networks [14, 13, 36, 17]) and one-stage networks (e.g., YOLO and SSD series networks [34, 35, 6, 32]). The major difference between two-stage and one-stage networks is on whether bounding box prediction and classification are separate or not. In the head part, two-stage networks find bounding boxes first and then classify objects in the bounding boxes. On the other hand, one stage networks predict bounding boxes and class probabilities together.

In general, one-stage networks are more compact than two-stage networks. For example, YOLO [34] with GoogLeNet [37] consists of 26 layers while Faster R-CNN with FPN [30] and ResNet-50 [18] consists of 213 layers in total. Since one-stage networks are more compact and faster than two-stage networks, most autonomous driving systems adopt one-stage networks like YOLO and SSD. For example, Autoware [24], a popular open-source autonomous driving system, allows to use either YOLO or SSD for object detection. Another open-source system, Apollo [1] also uses Apollo-OD for object detection which originates from YOLO networks.



(a) General architecture. **(b)** Computational cost. **(c)** Accuracy.

Figure 3 General architecture of existing object detectors and computational cost and accuracy of an existing object detector [6] with different image scales.

2.3 Dynamic Image Scaling

Dynamic image scaling is one of the optimization techniques that can accelerate object detection. It can enhance object detection speed by reducing the scale of the input image dynamically. Figure 3 shows how the computational cost changes as the image size changes on top of the existing deep learning framework [11]. Note that we downsample the input image to have the widths on the x-axis. In the graph, BFLOPS indicates the number of floating-point operations in billions. The graph shows that the computational cost decreases almost linearly as the image size decreases.

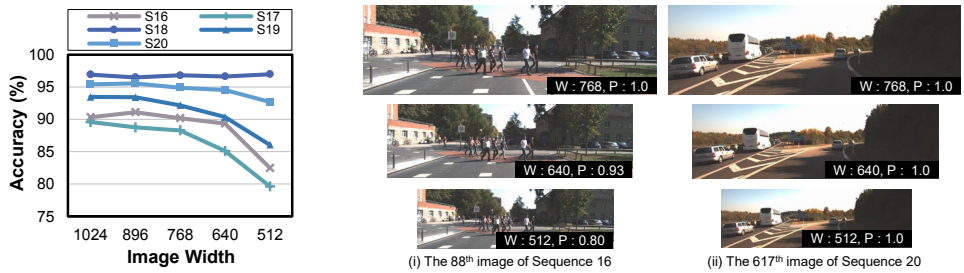
However, downsampling an image often incurs accuracy loss. Figure 3 also shows how the accuracy changes as the image size changes for the KITTI MOT dataset [12]. Here, we use mean average precision with IoU threshold $= 0.5$ (mAP_{50}) as the accuracy metric, which is commonly used to evaluate object detectors. The graph shows that the accuracy of the object detector tends to decrease as the image size decreases.

The interesting observation is that each image has a different sensitivity to image scaling in terms of accuracy. In other words, some images barely lose accuracy in downsampling while other images do not. In this work, *sensitivity* indicates how much image downsampling affects object detection accuracy for an image. If an image is highly sensitive to downsampling, it implies that the image would lose accuracy a lot in downsampling. This work will formally define the sensitivity in Section 4.

Figure 4a shows how the accuracy for each image sequence changes as the image size changes for the KITTI MOT dataset. The graph shows that some image sequence (S18) loses almost no point, but another image sequence (S17) loses 9.95 points at the minimum scale. Therefore, this work considers the images in S17 are more sensitive to image scaling than the images in S18. Thus, it is necessary to consider the sensitivity to image downsampling to reduce accuracy loss.

This work also observes that the sensitivity of an image differs according to the features of the image. Figure 4b is the collection of the sample images from the KITTI MOT dataset. The first image from S16 seems more complicated than the second image from S20. Whereas the first image contains objects that are difficult to detect (e.g., pedestrians on the road), the second image only contains simple objects (e.g., cars on the road). The difference may result in different sensitivities of S16 and S20. Since S16 has more complicated images than S20, S16 shows higher sensitivity than S20 as Figure 4a shows. Therefore, as the features of an image affect its sensitivity, we can infer the sensitivity based on the image.

Prior work [7] designs a dynamic image scaling scheme to enhance the accuracy of object detection, but the approach is not aware of real-time constraints nor sensitivity. It only predicts an optimal scale for a given image from a single image stream regardless of real-time



(a) Accuracy. **(b)** Sample images.

Figure 4 Object detection accuracy with different image scales for each image sequence in the KITTI MOT validation dataset.

constraints. The approach cannot find the optimal scales for multiple image streams that respect the time constraint. Therefore, the existing approach is not suitable for real-time object detection with multiple image streams.

This work proposes RTScale, which enables real-time object detection with a new adaptive image scaling scheme considering the scale sensitivity of multiple input images. RTScale predicts the sensitivity of each image and determines the appropriate scales of images based on real-time constraints and sensitivities. In this way, RTScale can reduce accuracy loss in image downsampling while satisfying real-time constraints.

3 Problem Statement

When an autonomous car drives, the car receives N images from its N cameras in a fixed interval. When the images arrive, the object detector processes the images with M processing units and makes decisions based on the results. Let $I_{i,j}$ denote the image frame of the j-th camera at the i-th interval. Then, at the i-th interval, the object detector conducts object detection tasks $\tau_{i,1}, ..., \tau_{i,N}$ for $I_{i,1}, ..., I_{i,N}$. To make a timely decision, the object detector must finish processing the N images within deadline D_i at each interval. Note that the relative deadline can differ across intervals depending on dynamic execution environment.

Before processing the images, the scheduler determines the scales of the images and schedules the object detection tasks for the images. In this paper, we define an image scale as (w, h) where w and h are the width and height of the image. The object detector maintains the set of scales S and chooses an appropriate scale from the set for each image. In other words, at the i-th interval, the scheduler determines $s_{i,j}$ as one of the scales in S for $j = 1, ..., N$ where $s_{i,j}$ denotes the image scale of $I_{i,j}$. The reason for assuming the predefined set of scales is that handling arbitrary image scales would incur considerable resizing overhead in practice because the system needs to configure the network layers every time if the input shape is arbitrary.

This work formulates the problem as follows.

Problem Statement. The problem is to find the optimal scales of input images, $\{s_{i,j}\}_{j\in[1,N]}$, and schedule the object detection tasks for the images, $\{\tau_{i,j}\}_{j\in[1,N]}$, at each interval i while satisfying the following constraint.

Deadline Constraint. The object detection tasks for the images finish before the deadline, i.e., for each $j = 1, ..., N$,

$$f_{i,j} \leq r_{i,j} + D_i$$

where $r_{i,j}$ and $f_{i,j}$ are the release time and completion time of $\tau_{i,j}$, respectively.

Table 1 Notation for the problem.

No.	Description	No.	Description
N	Number of cameras	$s_{i,j}$	Scale of $I_{i,j}$ for processing ($s_{i,j} \in S$)
M	Number of processing units	$\tau_{i,j}$	Object detection task for $I_{i,j}$
S	Set of scales to choose from	$r_{i,j}$	Release time of $\tau_{i,j}$
D_i	Relative deadline at the i-th interval	$f_{i,j}$	Completion time of $\tau_{i,j}$
$I_{i,j}$	The j-th image at the i-th interval	$\rho_{i,j}$	Scale sensitivity of $I_{i,j}$
$T[k]$	Latency of processing an image at $S[k]$		

The ultimate goal of this work is to maximize object detection accuracy while satisfying the deadline constraint. If we only consider the deadline constraint, forcing a very low scale to the images could be a solution. However, it is not desirable because aggressive image downsampling can cause a huge accuracy loss. Since each image has a different sensitivity to image scaling as shown in Section 2, it is essential to minimize accuracy loss by considering the sensitivity.

This work uses four assumptions for the problem: (i) The frame interval is longer than or equal to the relative deadline. (ii) The worst-case execution time of processing an image at a scale is given. Based on prior work [5, 20, 21, 19], it is possible to estimate the worst-case execution time of an object detector. Especially, [19] presents a layer-level worst-case execution time model for general neural networks. Therefore, we can obtain the worst-case execution time of the target object detector by combining the estimation models of its network layers. (iii) Every image is equally important in terms of autonomous driving as in Autoware [24]. In other words, each image provides equally meaningful information for autonomous driving. (iv) The time to schedule the object detection tasks is negligible compared with the time to detect objects in images. To justify the fourth assumption, we measure the scheduling overhead in the experiment and observe that scheduling takes less than $10^{-5}\%$ of the total execution time as shown in Section 5.2.4.

4 Design

While most existing object detectors process images at a fixed scale, RTScale provides sensitivity-aware adaptive image scaling for real-time object detection. Figure 5 briefly illustrates the overall object detection process with RTScale. RTScale extends an existing object detector with a new lightweight sensitivity inference module that infers the scale sensitivity of an input image. Since the sensitivity inference module reuses the features extracted by the object detector, the module can predict scale sensitivity with a few convolution layers only. While offline, RTScale trains the sensitivity inference module with the ground-truth scale sensitivities of training datasets. While online, RTScale determines the scales of the next images based on the real-time constraints and the sensitivity prediction results of the current images.

In this paper, we assume a YOLO-series object detector [6] as a baseline object detector because it is one of the most popular object detectors in open-source autonomous driving platforms [24, 25, 1]. However, note that the proposed method can apply to any object detector as well because it is based on the common characteristics of ordinary existing object detectors.

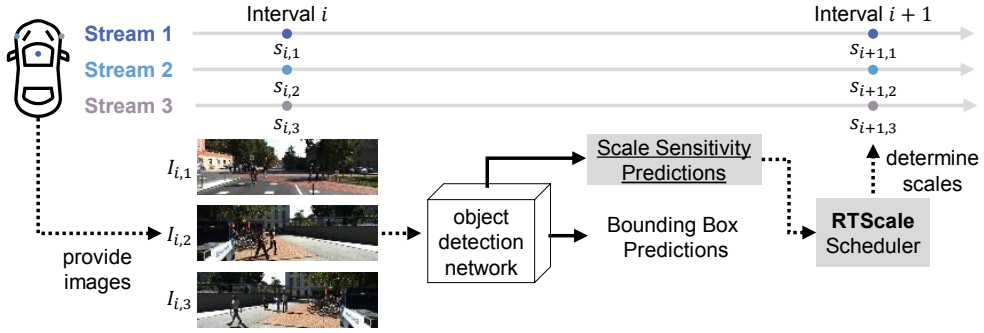


Figure 5 Overall object detection process with RTScale.

4.1 Scale Sensitivity Inference

This work defines *scale sensitivity* of an image ρ as the ratio of the accuracy obtained at the maximum scale to the accuracy obtained at the minimum scale, i.e.,

$$\rho = \frac{A[s^{max}]}{A[s^{min}]} \tag{1}$$

where $A[s]$ indicates the object detection accuracy of the image when detecting objects at a scale s, and s^{max} and s^{min} are the maximum and minimum scales in S, respectively.

The meaning of scale sensitivity is the accuracy loss ratio at a minimum scale. If an image loses accuracy a lot at a minimum scale, the image will have a high scale sensitivity. On the other hand, if an image loses little accuracy at a minimum scale, the image will have a low scale sensitivity. It is possible for a scale sensitivity to be smaller than one because object detectors sometimes can detect objects better at a lower scale. For example, if an image contains a very large object like a train, it may be better to process the image at a lower scale [7].

The scale sensitivity definition presupposes a monotonic relationship between image scale and object detection accuracy. This work has tried different metrics for the sensitivity to reflect the non-monotonic relationship between image scale and object detection accuracy as shown in Figure 4a. However, the sensitivity inference problem becomes too complex for the inference module to be trained. To simplify the sensitivity inference module, this work assumes that image scale and object detection are in a monotonic relationship.

Network Extension. This work extends an existing object detector to infer the scale sensitivity of an image along with bounding box prediction and classification. This work designs the scale sensitivity inference module to exploit the rich features from the existing object detector for scale sensitivity inference. As explained in Section 2, ordinary object detectors have a backbone network to extract the features from an input image and use the features to infer bounding box locations and categories. Since the sensitivity inference module also needs distinct features from the same input image, this work reduces the complexity of the sensitivity inference module by sharing the feature maps from the backbone network.

Table 2 describes the detailed architecture of the sensitivity inference module. The module applies two pairs of 3×3 and 1×1 convolution layers with the Leaky ReLU activation function [33] to the feature maps from the backbone network. Next, the module applies (global) average pooling to the output feature maps and obtains 512 features per image. Here, the global average pooling enables the module to get the same size of features for different image scales. Then, the module obtains the final sensitivity prediction by applying a dense layer with the linear activation function.

Table 2 The architecture of sensitivity inference module (Input: Feature maps from the backbone network of which shape is $(N, 1024, H, W)$, Output: Normalized scale sensitivity).

Type	Kernel	Padding	Activation	Output Dim.
Convolution	3×3	1	Leaky	$(N, 512, H, W)$
Convolution	1×1	0	Leaky	$(N, 512, H, W)$
Convolution	3×3	1	Leaky	$(N, 512, H, W)$
Convolution	1×1	0	Leaky	$(N, 512, H, W)$
Average pooling	Global	-	-	$(N, 512, 1, 1)$
Dense	-	-	Linear	(N)

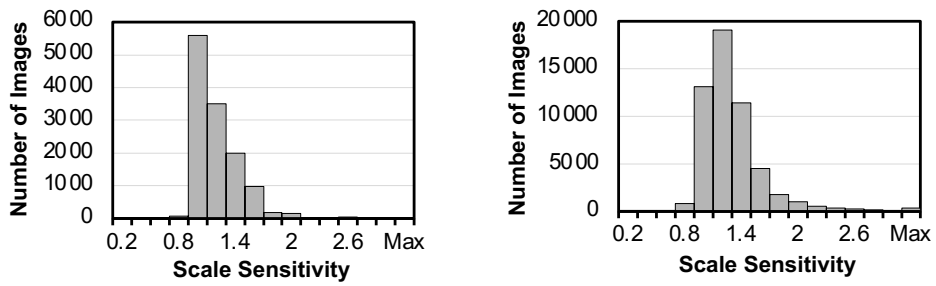
Ground-truth Label Generation. To train the scale sensitivity module, this work calculates the ground-truth sensitivities of the images from train datasets. For the minimum and maximum scales in S, this work records the object detection accuracy of the baseline object detector for each image in the train datasets. Then, this work calculates the ground-truth sensitivity of each image using (1).

This work uses the F1 score as the accuracy metric for scale sensitivity calculation to consider both precision and recall. Precision is insufficient to evaluate the accuracy for a *single* image because precision may favor low-resolution images with a smaller number of *valid* predictions (i.e., the total number of true positives and false positives of which confidence scores are larger than a threshold). In general, the total number of valid predictions tends to decrease as the image scale decreases. Thus, if there are few valid predictions of an image, its precision becomes undesirably high. Therefore, it is necessary to consider recall also to mitigate the problem. Note that mean average precision (mAP) is not applicable here because it is an accuracy metric for an entire dataset, not for a single image.

$$A[s] = \frac{2}{\frac{1}{pr[s]} + \frac{1}{rc[s]}} \tag{2}$$

Equation 2 is the definition of the F1 score where $pr[s]$ and $rc[s]$ denote the precision and recall of an image obtained at a scale s, respectively. In object detection, we regard a bounding box prediction is true positive if the predicted bounding box overlaps with a true bounding box with IoU > 0.5 and the predicted category is same as the category of the true bounding box. Here, IoU is the abbreviation of "Intersection over Union", which indicates the ratio of the overlap area of two bounding boxes to the total area of two bounding boxes.

To facilitate deep learning, this work normalizes the ground-truth sensitivity values with (3). In the equation, $\hat{\rho}^{min}$ and $\hat{\rho}^{max}$ indicate the minimum and maximum scale sensitivities, respectively. Note that $\hat{\rho}^{min}$ and $\hat{\rho}^{max}$ depend on the train datasets. Figure 6 shows the



(a) KITTI MOT. **(b)** BDD100K MOT.

Figure 6 Scale sensitivity distribution of each driving dataset.

scale sensitivity distributions of two driving datasets, KITTI MOT [12] and BDD100K MOT [43]. The graphs show that BDD100K MOT has a broader distribution of scale sensitivity than KITTI MOT. Thus, this work uses a smaller $\hat{\rho}^{min}$ and a larger $\hat{\rho}^{max}$ for BDD100K MOT compared with KITTI MOT. In inference, this work denormalizes the predicted scale sensitivity for use in scheduling.

$$\rho^{norm} = (\rho - \hat{\rho}^{min})/(\hat{\rho}^{max} - \hat{\rho}^{min}) \tag{3}$$

Training: This work regards scale sensitivity inference as a linear regression problem and trains the sensitivity inference module using a smooth L1 loss function with the ground-truth labels. Equation 4 is the definition of the loss function where y is the predicted sensitivity and ρ^{norm} is the normalized ground-truth sensitivity. This work accumulates the loss for each image when the mini-batch size is larger than one.

$$L(y, \rho^{norm}) = \begin{cases} 0.5 \times (y - \rho^{norm})^2 & \text{if } |y - \rho^{norm}| < 1 \\ |y - \rho^{norm}| - 0.5 & \text{otherwise} \end{cases} \tag{4}$$

4.2 Scheduling

The RTScale scheduler determines the scales of the next images considering both the real-time constraint and the sensitivity prediction of the current images. It is based on the assumption that two consecutive frames from the same image stream have similar image features. That is, two consecutive images in the same image stream would have similar scale sensitivities. We consider the assumption is reasonable because prior work on video object detection [48, 49, 47, 46] is also based on the assumption.

The basic idea for determining the scales is to minimize the expected accuracy loss relative to the accuracy obtained at the maximum scale. This work calculates the relative accuracy loss of a scale with scale sensitivity. Let $Q(k_1, k_2)$ be the ratio of the k_2-th smallest scale over the k_1-th smallest scale in S. For example, $Q(1, |S|)$ is the ratio of the maximum scale over the minimum scale in S. According to the definition,

$$Q(1, |S|) = \Pi_{k \in [1, |S|-1]} Q(k, k+1)$$

Then, this work defines the expected accuracy gain of the k-th smallest scale relative to the minimum scale as follows:

$$gain(k, \rho) = \rho^{\log_{Q(1,|S|)} Q(1,k)} \tag{5}$$

Here, if $Q(1, 2) \simeq ... \simeq Q(|S| - 1, |S|)$, we can simplify (5) as follows:

$$gain(k, \rho) = \rho^{\frac{k-1}{|S|-1}} \tag{6}$$

Finally, we define the expected loss of the k-th smallest scale relative to the maximum scale as follows:

$$loss(k, \rho) = \begin{cases} \frac{gain(|S|, \rho)}{gain(k, \rho)} = \rho^{\frac{|S|-k}{|S|-1}}, & \text{if } k \geq 1 \\ \infty, & \text{otherwise} \end{cases} \tag{7}$$

This work designs a scheduling algorithm that gradually finds the scale and schedule of each image that meet the time constraint for M (identical) processing units. After processing $I_{i,1}, I_{i,2}, ..., I_{i,N}$ at the i-th interval, the object detector invokes the scheduler with the sensitivity predictions of the images. The algorithm takes the set of sensitivities

Algorithm 1 RTScale scheduling algorithm.

Input : Sensitivities of the previous frames $\rho_1, ..., \rho_N$
Relative deadline D
Output : Schedule of the next frames SC

1 **if** $\lceil N/M \rceil \cdot T[1] > D$ **then**
2 Return with an error
3 **end**
4 $SC \leftarrow$ `initialize`$(\{\rho_j\}_{j \in [1,N]})$
5 **while** $SC.makespan > D$ **do**
6 $j^* \leftarrow \arg\min_j loss(SC[j].scale - 1, SC[j].sensitivity)$
7 $SC[j^*].scale \leftarrow SC[j^*].scale - 1$
8 SC.`update`(j^*)
9 **end**
10 SC.`optimize`()

Algorithm 2 `initialize`$(\rho_1, \rho_2, ..., \rho_N)$.

Input : Scale sensitivities $\rho_1, ..., \rho_N$
Output : Initial schedule SC

1 **for** $j \in [1, N]$ **do**
2 $SC[j].id \leftarrow j$
3 $SC[j].sensitivity \leftarrow \rho_j$
4 $SC[j].scale \leftarrow |S|$ `// Maximum scale`
5 **end**
6 $SC \leftarrow$ Sort SC in descending order of sensitivity
7 $W \leftarrow \{0, 0, ..., 0\}$ `// Workload of each unit`
8 **for** $j \in [1, N]$ **do**
9 $p^* \leftarrow j \mod M$
10 $SC[j].proc \leftarrow p^*$
11 $SC[j].track \leftarrow W$
12 $W[p^*] \leftarrow W[p^*] + T[|S|]$
13 **end**
14 $SC.makespan \leftarrow \max_{p \in [1,M]} W[p]$

$\rho_{i,1}, \rho_{i,2}, ..., \rho_{i,N}$ and the relative deadline for the next interval D_{i+1} as its inputs. Considering both the sensitivities and the relative deadline, the algorithm determines the scales of the next images $s_{i+1,1}, s_{i+1,2}, ..., s_{i+1,N}$ and the assignment of processing units.

Algorithm 1 is the pseudo code of the RTScale scheduling algorithm. First, the algorithm checks whether the given images are schedulable or not by comparing the minimum possible *makespan* and the relative deadline (Line 1 to Line 3). After the test, the algorithm generates an initial schedule setting all the image scales as the maximum scale (Line 4). Then, the algorithm gradually lowers the scales until the makespan does not exceed the given relative deadline (Line 5 to Line 9). At every iteration, the algorithm finds the scale with the minimum expected loss, lowers the scale, and updates the schedule to reflect the change. Note that the algorithm assumes that S is sorted in advance.

Algorithm 2 and Algorithm 3 show how to find the minimum makespan schedule in detail. Since the minimum makespan scheduling problem is known as strongly NP-hard [40], the algorithms are based on a 4/3 approximation algorithm which sorts the set of tasks in descending order of latency and greedily assigns each task to the processing unit with the minimum workload. Rather than calculating the entire minimum makespan schedule at every iteration, this work optimizes the algorithms to incrementally update the minimum

Algorithm 3 SC.`update`(j^*).

```
   Input  : Index of the target image j*
1  W ← SC[j*].track
2  for j ∈ [j*, N] do
3  |  p* ← arg min_p W[p]
4  |  SC[j].proc ← p*
5  |  SC[j].track ← W
6  |  W[p*] ← W[p*] + T[SC[j].scale]
7  end
8  SC.makespan ← max_{p∈[1,M]} W[p]
```

makespan schedule. After sorted once in initialization, the order of the tasks remains the same as the main loop iterates. Therefore, the algorithms can simply update the necessary part of the schedule only (Line 2 and Line 7 in Algorithm 3).

After determining the schedule, Algorithm 1 optimizes the schedule as finalization. Since the algorithm reduces the scale of each image based on its expected accuracy loss, each processing unit may have spare time until the deadline. Therefore, the algorithm checks the amount of slack for each processing unit and increases the scale of each image if possible in the descending order of scale sensitivity.

The underlying principle of the algorithm is to lower image scales to satisfy the deadline constraint while minimizing its total accuracy loss. By reducing the scale of an image that has the minimal accuracy loss until satisfying the time constraint, the algorithm will obtain the maximal accuracy product of the scaled images that respects the time constraint.

The computation complexity of the scheduling algorithm is $O(N^2KM)$ where $K = |S|$. In the algorithm, the main loop for determining the next scales dominates the computation complexity of the algorithm. In the worst case that all images require the minimum scale, the outer loop iterates $N(K-1)$ times, and the computation complexity of the `update` function is $O(NM)$. Therefore, the computation complexity of the algorithm is $O(N^2KM)$.

5 Evaluation

5.1 Experimental Setup

This work implements RTScale on top of the Darknet deep learning framework from prior work [11]. This work extends the state-of-the-art object detector [6] with the scale inference module in Section 4.1. Since the original object detector is supposed to use a single image scale, this work modifies the deep learning framework to dynamically change the scales of input images as the scheduler directs. Furthermore, this work enables the deep learning framework to support multiple image streams and multiple processing units for inference.

To show the effectiveness of RTScale, this work compares different image scaling schemes on top of the framework:

- **AvgScale**: chooses the maximum scale for each processing unit that does not violate the deadline constraint, i.e., choose the maximum scale such that $N_p \cdot T[k] \leq D_i$ where N_p indicates the number of images assigned to the p-th processing unit.
- **RTScale-pred**: chooses the scales using the proposed scheduling algorithm based on scale sensitivity predictions.
- **RTScale-gt**: chooses the scales using the proposed scheduling algorithm with ground-truth scale sensitivities.

This work evaluates the image scaling schemes with two driving datasets, KITTI MOT [12] and BDD100K MOT [43]. The reason for using the multiple object tracking datasets is that existing object detection datasets do not provide multiple streams of consecutive image frames. Existing object detection datasets only provide key frames for training and testing. Therefore, this work uses the multiple object tracking datasets for evaluation. Note that the multiple object tracking datasets only consider movable objects such as cars and pedestrians, unlike ordinary object detection datasets.

Since the KITTI MOT dataset does not contain a validation set, this work divides the train set of the KITTI MOT dataset for training and validation. Among 21 image sequences of the train set, this work uses 16 sequences for training and the other five sequences for validation. In addition, this work supplements the small volume of the KITTI MOT train set with the KITTI object detection dataset. In the case of BDD100K MOT, this work samples one image frame every other five frames from the train set to avoid overfitting from having too many similar images.

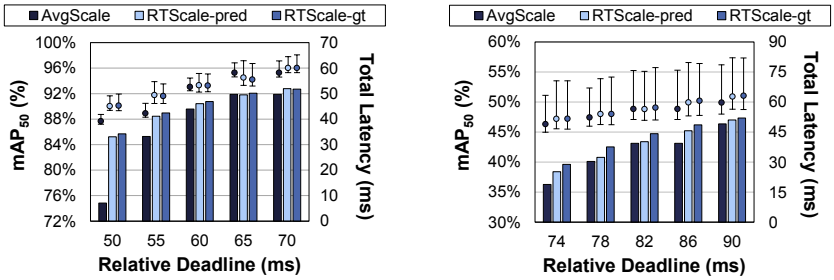
This work first trains the baseline object detector for each dataset with multiple image scales. The framework dynamically scales the input images by randomly choosing a scaling factor between $[1/1.4, 1.4]$ every 10 iterations. This work uses pretrained network parameters for the first 137 layers of the baseline object detector. This work uses 0.001 as the learning rate for the datasets and divides the rate by 10 after 80% and 90% of the total iterations as prior work [6].

After training the baseline object detector, this work trains the sensitivity inference module for each dataset. This work generates ground-truth sensitivity labels for each dataset and filters several outliers to facilitate deep learning. This work uses $(0.6, 2.8)$ and $(0.3, 4.0)$ as (ρ^{min}, ρ^{max}) in (3) to normalize the ground-truth sensitivity values for KITTI MOT and BDD100K MOT, respectively. For KITTI MOT, this work uses a larger learning rate considering the small volume of the KITTI MOT train set. Similar to the baseline detector, this work divides the learning rate by 10 after 80% and 90% of the total iterations.

For evaluation, this work generates three artificial image streams with the validation images of each dataset. This work divides a set of image sequences into three image streams. Each stream in KITTI MOT and BDD100K MOT has 693 and 11,329 images, respectively. For evaluating with multiple processing units, this work further divides three image streams from KITTI MOT into six image streams. Although each image stream includes real-world road images, the image streams are not from the cameras on the same vehicle. It might not

Table 3 Datasets and training parameters.

Information		KITTI MOT	BDD100K MOT
Number of train images		12,900	55,616
Number of validation images		2,079	33,987
Original image size		1242 × 375	1280 × 720
Parameter		KITTI MOT	BDD100K MOT
Baseline	Number of iterations	16000	16000
	Batch size	64	64
	Learning rate	0.001	0.001
Module	Number of iterations	70000	70000
	Batch size	2	2
	Learning rate	0.001	0.0001
Image scales		1024 × 288, 896 × 256, 768 × 224, 640 × 192, 512 × 160	768 × 416, 704 × 384, 640 × 352, 576 × 320, 512 × 288



(a) KITTI MOT ($N = 3, M = 1$). **(b)** BDD100K MOT ($N = 3, M = 1$).

Figure 7 Object detection accuracy and total latency with a single processing unit (N: Number of image streams, M: Number of GPUs).

be ideal, but it is the best possible option because there is no available multi-camera driving dataset. This work conducts experiments with two Intel(R) Xeon(R) Silver 4210 CPUs and up to three NVIDIA GeForce RTX 2080 Ti GPUs using CUDA 10.0 and cuDNN 7.6.4.

5.2 Results

5.2.1 Accuracy and Latency

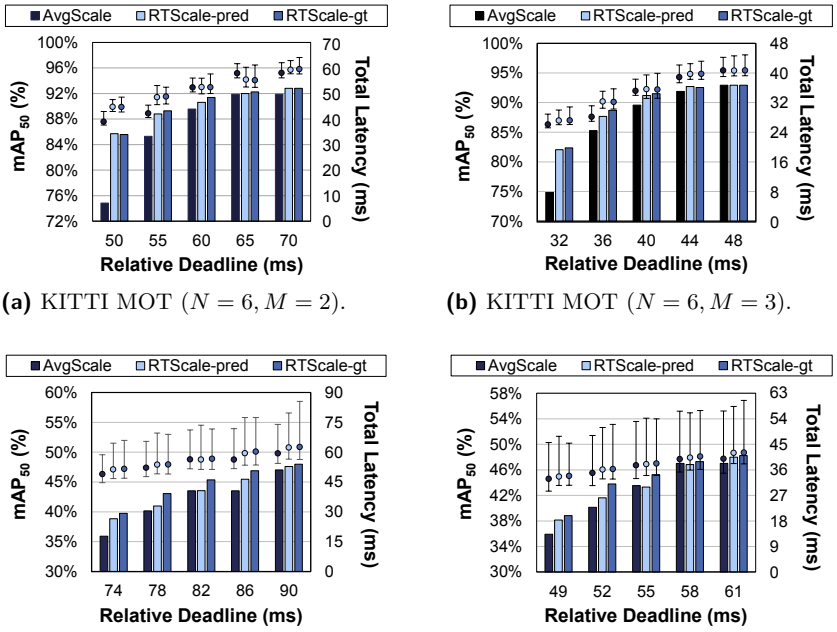
This work measures the accuracy and latency of the object detector for a given relative deadline with different image scaling schemes. This work evaluates object detection accuracy with the mean average precision metric with IoU threshold = 0.5 using the evaluation code of the Darknet framework. In addition, this work measures the total latency for processing all the images from image streams within an interval. This work determines relative deadlines considering the worst-case object detection latency of processing an image at each scale.

Figure 7 shows the accuracy and latency of the object detector for KITTI MOT and BDD100K MOT with one processing unit. In the figures, the rectangular bars indicate the accuracy of the object detector and the error bars indicate the range of object detection latency during all the intervals. For different relative deadlines, all the image scaling schemes meet the deadline constraints because they always choose the appropriate set of scales that would not violate the deadline constraint.

Overall, with the same relative deadline, RTScale-pred and RTScale-gt obtain better accuracy than AvgScale. As shown in the graphs, RTScale-pred and RTScale-gt enhance the object detection accuracy by 10.4 and 10.8 points at most. Since RTScale-pred and RTScale-gt determine the scales of the images considering scale sensitivity, RTScale-pred and RTScale-gt can reduce accuracy loss from image downsampling compared with AvgScale.

The amount of accuracy gain with RTScale tends to decrease as the relative deadline increases. It is because the accuracy gap between two similar scales tends to be smaller for the higher scales. In Figure 4a, the object detector obtains 5.48 more points for S17 with the image width of 768 than the image width of 640. On the other hand, the accuracy gap between the image widths of 1024 and 896 is only 0.83 points. The result implies that there are less opportunities to enhance accuracy through adaptive image scaling when the higher scales are available (i.e., when the relative deadline is large).

In general, RTScale-gt obtains better accuracy than RTScale-pred because RTScale-gt uses ground-truth scale sensitivities in scheduling. Therefore, RTScale-gt can better predict expected accuracy loss than RTScale-pred. In the experiment, RTScale-gt gains at most 1.8 more points in accuracy compared with RTScale-pred. Nevertheless, RTScale-pred performs almost as good as RTScale-gt in the experiments.



(a) KITTI MOT ($N = 6, M = 2$). **(b)** KITTI MOT ($N = 6, M = 3$).

(c) BDD100K MOT ($N = 6, M = 2$). **(d)** BDD100K MOT ($N = 6, M = 3$).

Figure 8 Object detection accuracy and total latency with multiple processing units (N: Number of image streams, M: Number of GPUs).

Figure 8 shows the accuracy and latency of the object detector for KITTI MOT and BDD100K MOT with two and three processing units. RTScale assigns each task to a certain processing unit regarding the scale sensitivity. For example, RTScale can have a processing unit dedicated to a highly sensitive image. Thus, RTScale can better utilize the multiple processing units to reduce accuracy loss than AvgScale. As shown in the graphs, RTScale-pred and RTScale-gt enhance the object detection accuracy by 10.9 and 10.7 points at most with the two processing units.

Interestingly, RTScale-pred outperforms RTScale-gt in some cases. It is because choosing the maximum scale is not always the best, even for the images with scale sensitivity larger than one. This work observes that detecting objects at a medium scale sometimes results in the best accuracy. Since RTScale calculates the scale sensitivity regarding the maximum and minimum scales only, RTScale sometimes fails to predict the expected accuracy loss correctly. However, predicting the non-monotonic tendency of accuracy change is too complicated for deep neural networks to learn. Therefore, RTScale only considers the monotonic tendency of accuracy change.

Furthermore, RTScale may not perform to the best because the datasets only provide a few frames per second. Although the object detector can process three images within 100 ms, the actual time interval between two consecutive images in the datasets is much longer. It means that two images may not be similar to each other. It can hinder RTScale from performing to the best because the scale sensitivities of the two images may not be similar to each other, which is different from our assumption.

5.2.2 Dynamic Deadline Adaptation

This work evaluates how the object detector can well adapt to dynamically changing relative deadline with the KITTI MOT dataset. This work randomly generates the sequence of relative deadlines and provides a different relative deadline to the object detector at every

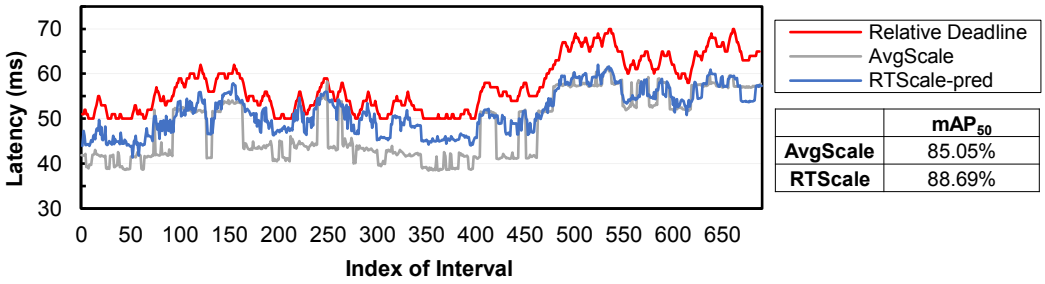
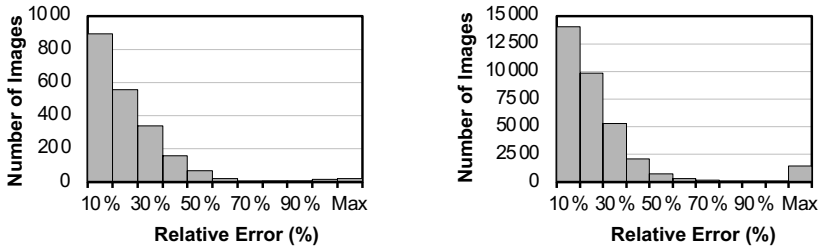


Figure 9 Object detection accuracy and latency when changing the relative deadline at every interval.



(a) KITTI MOT. **(b)** BDD100K MOT.

Figure 10 Relative error distribution of scale sensitivity predictions.

interval. This work randomly chooses an integer within $[50, 70]$ for a relative deadline. Here, since each image stream of the KITTI MOT dataset contains 693 images, the total number of intervals is 693, accordingly.

Figure 9 shows how the latency of the object detector changes according to the relative deadline. In the figure, the red line indicates the relative deadline at each iteration, the grey line indicates the latency of the object detector with AvgScale, and the blue line indicates the latency of the object detector with RTScale-pred. Figure 9 also provides the accuracy of the object detector with AvgScale and RTScale-pred.

The evaluation result shows that RTScale better adapts to the deadline compared with AvgScale because RTScale applies different scales for the images. Although the two image scaling schemes enable the object detector to meet the deadline constraint at every interval, the latency with RTScale changes more smoothly according to the given deadline. Furthermore, RTScale obtains 3.64 points higher object detection accuracy compared with AvgScale. The result shows that RTScale can well adapt to the dynamic deadline while enhancing object detection accuracy.

5.2.3 Scale Sensitivity Inference

This work also evaluates how accurately the scale sensitivity inference module can predict scale sensitivity. This work calculates the relative errors of the predicted scale sensitivity with the ground-truth scale sensitivity for the validation set. Figure 10a and Figure 10b are the histograms that show the distributions of relative errors for KITTI MOT and BDD100K MOT. On average, the scale sensitivity module obtains 17% and 21% relative errors for KITTI MOT and BDD100K MOT, respectively.

(i) **Relative Deadline**: 50 ms (ii) **Relative Deadline**: 60 ms

Figure 11 Sample object detection results with the image scales used in detection (TP: True Positive, FP: False Positive).

Figure 7 clearly shows the consequence of inaccurate scale sensitivity inference. Since the sensitivity module relatively performs worse for BDD100K MOT than KITTI MOT, the gap between RTScale-pred and RTScale-gt is larger for BDD100K MOT. While the largest gap between RTScale-pred and RTScale-gt is 0.5 points for KITTI MOT, the largest gap between RTScale-pred and RTScale-gt is 1.8 points for BDD100K MOT. It implies that the errors in scale sensitivity predictions can degrade accuracy.

However, the relative errors of scale sensitivity predictions do not directly cause accuracy degradation because the scheduler determines the scales of images by comparing the expected accuracy losses of the images. That is, the scheduler considers the relative differences in their sensitivities. Therefore, if the predicted sensitivities of the images show a similar difference to the ground-truth sensitivities, RTScale-pred still can find an effective solution as RTScale-gt. As a result, RTScale-pred could obtain a comparable accuracy with RTScale-gt in the experiments.

This work also visualizes the object detection results of sample images from the KITTI MOT to show how the quality of object detection results differs with image scales. Figure 11 shows the object detection results of the same images with different relative deadlines. In the figure, each image includes the image scale used in object detection. For readability, this work resizes the images to have the same size. In addition, the color of a bounding box indicates the category of the object. Overall, the object detection results with a high scale have more true positives than the results with a lower scale. For example, in the case of the middle image in the figure, the object detector fails to detect a hidden car on the right side at the lowest scale.

5.2.4 Memory and Scheduling Overhead

This work measures the system-level memory and scheduling overhead compared with the original object detection system. RTScale requires extra memory because the system needs to store the parameters for the scale sensitivity inference module and maintain the multiple descriptors for each convolution layer to avoid the resizing overhead. However, the total memory overhead is at most 2.10 % only compared with the amount of memory that the original system uses. Furthermore, RTScale incurs little scheduling overhead with the small N and M compared with the total latency of processing images.

Table 4 Memory and scheduling overhead of RTScale.

Configuration	$N = 3, M = 1$	$N = 6, M = 2$	$N = 6, M = 3$
Memory Overhead	0.32 %	1.54 %	2.10 %
Scheduling Overhead	$< 10^{-5}$ %	$< 10^{-5}$ %	$< 10^{-5}$ %

6 Related Work

6.1 Real-time Object Detection

Previous work [22, 19, 38] has studied real-time object detection to finish object detection within a given deadline.

Jang et al. [22] design a real-time object detector for autonomous driving considering the end-to-end delay of object detection. Jang et al. observe that existing object detectors suffer from severe time lags, even excluding inference time. Jang et al. address that the time lags come from queuing delay, pipeline imbalance, and resource contention (especially on integrated CPU-GPU platforms). To reduce the end-to-end delay of an object detector, Jang et al. propose three optimization techniques: on-demand capture, zero-slack pipeline, and contention-free pipeline. Since the prior work targets to optimize end-to-end delay rather than inference time, this work can apply RTScale to R-TOD to further optimize end-to-end object detection latency, including inference time.

S^3DNN [45] is a system solution for executing multiple DNN workloads in real time. S^3DNN guarantees real-time performance with two main techniques: (i) system-level data fusion and (ii) supervised streaming and scheduling. First, S^3DNN fuses multiple DNN workloads into one DNN instance to enhance resource utilization within the memory limit. Second, S^3DNN enables streaming processing of multiple GPU kernels from different DNN instances and schedules the kernels considering concurrency benefits. The main objective of S^3DNN is to enhance the throughput of DNN workloads, which might degrade pipeline efficiency in autonomous driving. Nevertheless, this work can employ its techniques to optimize throughput because the techniques are orthogonal to the approach of this work.

Heo et al. [19] propose multi-path neural networks that can adapt to time-varying time constraints by dynamically changing their execution paths. According to the time constraints, the multi-path neural network can skip layers, generate a different number of region proposals, and switch to another branch. Heo et al. also provide the worst-case execution time model for deep neural networks considering the worst-case memory contention. Although the prior work shows that the multi-path neural network can well adapt to the time constraints, it requires designing an elaborate multi-path neural network to minimize accuracy loss.

IntPred [38] is an object detection scheme that reduces computational cost with an interpolation prediction. Since objects move slowly across consecutive frames, IntPred only runs an object detector for a partial set of image frames. Based on the interpolation prediction, IntPred adjusts the location of the objects for subsequent image frames. By avoiding processing every image frame, IntPred can reduce object detection time and power consumption. However, it can be risky to skip image frames because objects can suddenly appear in the skipped frames, especially in the street images that change dynamically.

6.2 Real-time DNN Inference

Rather than focusing on object detection, other work [42, 26, 31, 27, 23, 41, 3, 4] proposes general real-time DNN frameworks.

Recent work [42, 26, 31] has proposed to use the concept of *early exiting* for real-time DNN inference, which allows a neural network to generate the output early. Yao et al. [42] suggest controlling the number of network stages with early exiting to provide intelligent real-time edge services. Liu et al. [31] extend the previous work [42] to consider the criticality of data within a scene. Kim et al. [26] propose a hierarchical neural network that can provide abstract classifications before final concrete classification. Similar to RTScale, the recent work exploits the architecture of existing networks to provide real-time inference. However, the work requires designing a network with multiple exits in advance for dynamic latency adjustment. Therefore, the network cannot easily adapt to different run-time environments.

Lee et al. [27] introduce SubFlow, which enables real-time inference and training by dynamically pruning neurons. SubFlow dynamically generates subgraphs according to dynamic time constraints. By ranking neurons of each layer considering their importance, SubFlow can find the best subgraph that satisfies a given time constraint. Lee et al. propose time-bound inference and training of convolutional and fully-connected layers. Since its dynamic model compression approach is orthogonal to adaptive image scaling, RTScale is applicable in combination with SubFlow. However, SubFlow only supports convolution and fully-connected layers for now, while most object detectors include other types of layers.

DART [41] is a pipeline-based DNN scheduling framework, which provides a deterministic response time for processing multiple DNN models. DART minimizes the response time using data-level parallelism, allocating tasks into multiple CPUs and GPUs. DART utilizes two types of data-level parallelism, inter-node pipelining and intra-node data parallelism, to overcome the resource limitation of local accelerators and exploit multiple processing units efficiently. DART provides a time-predictable DNN execution for multiple processing units.

DeepRT [23] and PredJoule [4] utilize DVFS (Dynamic Voltage-Frequency Scaling) to satisfy time constraints and optimize energy consumption in neural network execution. They dynamically change the DVFS configuration according to time constraints and energy consumption. DeepRT also employs dynamic model compression to reduce the computational cost of executing deep neural networks on mobile devices. PredJoule finds the optimal DVFS configuration considering the performance and energy characteristics of different layers.

ApNet [3] is a timing-predictable runtime system for DNN workloads, which applies approximation to neural networks to satisfy real-time requirements. ApNet chooses an appropriate approximation strategy for each layer based on how the resource utilization of the target device changes with different approximation strategies. In addition, ApNet designs a runtime system that can enhance resource sharing and concurrency via approximation.

Although the existing approaches [23, 4, 3] for real-time DNN inference allow each inference task to finish within a deadline, the approaches either require the system support or incur a relatively large accuracy loss.

7 Conclusion

This work proposes RTScale, which enables real-time object detection through adaptive image scaling while minimizing accuracy loss. Based on the observation that each image has a different sensitivity to image scaling with regard to object detection accuracy, RTScale finds appropriate scales for images from multiple streams considering both scale sensitivity and real-time constraint. RTScale enables existing object detectors to infer the sensitivity by adding a few layers for sensitivity inference. This work evaluates RTScale with other image scaling schemes with two popular driving datasets. The evaluation results show that RTScale can meet real-time constraints with minimal accuracy loss.

References

1 Apollo. https://apollo.auto/index.html.

2 Autoware.ai Core Perception Github Repository. https://github.com/Autoware-AI/core_perception, May 2021.

3 Soroush Bateni and Cong Liu. ApNet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018. doi:10.1109/RTSS.2018.00017.

4 Soroush Bateni, Husheng Zhou, Yuankun Zhu, and Cong Liu. PredJoule: A timing-predictable energy optimization framework for deep neural networks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, December 2018. doi:10.1109/RTSS.2018.00020.

5 Adam Betts and Alastair Donaldson. Estimating the wcet of gpu-accelerated applications using hybrid analysis. In *2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*, 2013. doi:10.1109/ECRTS.2013.29.

6 Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection, 2020. doi:10.48550/arXiv.2004.10934.

7 Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. AdaScale: Towards real-time video object detection using adaptive scaling. In *Proceedings of Machine Learning and Systems 2019*, pages 431–441. 2019.

8 Ting-Wu Chin, Chia-Lin Yu, Matthew Halpern, Hasan Genc, Shiao-Li Tsao, and Vijay Janapa Reddi. Domain-specific approximation for object detection. *IEEE Micro*, 38(1):31–40, 2018. doi:10.1109/MM.2018.112130335.

9 Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, Seiya Maeda, Takuya Azumi, Kenji Funaoka, and Shinpei Kato. Towards heterogeneous computing platforms for autonomous driving. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019. doi:10.1109/ICESS.2019.8782446.

10 Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

11 Darknet. https://github.com/AlexeyAB/darknet.

12 Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. doi:10.1109/CVPR.2012.6248074.

13 Ross Girshick. Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. doi:10.1109/ICCV.2015.169.

14 Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014. doi:10.1109/CVPR.2014.81.

15 Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15. JMLR.org, 2015.

16 Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, Cambridge, MA, USA, 2015. MIT Press.

17 Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, October 2017. doi:10.1109/ICCV.2017.322.

18 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi:10.1109/CVPR.2016.90.

19 Seonyeong Heo, Sungjun Cho, Youngsok Kim, and Hanjun Kim. Real-time object detection system with multi-path neural networks. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020. doi:10.1109/RTAS48715.2020.000-8.

20 Vesa Hirvisalo. On static timing analysis of gpu kernels. In *14th International Workshop on Worst-Case Execution Time Analysis*, OpenAccess Series in Informatics (OASIcs), 2014.

21 Yijie Huangfu and Wei Zhang. Static wcet analysis of gpus with predictable warp scheduling. In *2017 IEEE International Symposium on Real-Time Computing (ISORC)*, 2017. doi:10.1109/ISORC.2017.24.

22 Wonseok Jang, Hansaem Jeong, Kyungtae Kang, Nikil Dutt, and Jong-Chan Kim. R-TOD: Real-time object detector with minimized end-to-end delay for autonomous driving. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 191–204, 2020. doi:10.1109/RTSS49844.2020.00027.

23 Woochul Kang and Jaeyong Chung. DeepRT: Predictable deep learning inference for cyber-physical systems. *Real-Time Systems*, 55(1):106–135, January 2019. doi:10.1007/s11241-018-9314-y.

24 Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, 2015. doi:10.1109/MM.2015.133.

25 Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, pages 287–296. IEEE Press, 2018. doi:10.1109/ICCPS.2018.00035.

26 Jung-Eun Kim, Richard Bradford, Man-Ki Yoon, and Zhong Shao. ABC: Abstract prediction before concreteness. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1103–1108, 2020. doi:10.23919/DATE48585.2020.9116479.

27 Seulki Lee and Shahriar Nirjon. SubFlow: A dynamic induced-subgraph strategy toward real-time dnn inference and training. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020. doi:10.1109/RTAS48715.2020.00-20.

28 Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

29 Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E. Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18. Association for Computing Machinery, 2018. doi:10.1145/3173162.3173191.

30 Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi:10.1109/CVPR.2017.106.

31 Shengzhong Liu, Shuochao Yao, Xinzhe Fu, Huajie Shao, Rohan Tabish, Simon Yu, Ayoosh Bansal, Heechul Yun, Lui Sha, and Tarek Abdelzaher. Real-time task scheduling for machine perception in in intelligent cyber-physical systems. *IEEE Transactions on Computers*, 2021. doi:10.1109/TC.2021.3106496.

32 Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*. Springer International Publishing, 2016. doi:10.1007/978-3-319-46448-0_2.

33 Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

34 Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi:10.1109/CVPR.2016.91.

35 Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. doi:10.1109/CVPR.2017.690.

36 Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 2017. doi:10.1109/TPAMI.2016.2577031.

37 Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. doi:10.1109/CVPR.2015.7298594.

38 Hamid Tabani, Matteo Fusi, Leonidas Kosmidis, Jaume Abella, and Francisco J. Cazorla. IntPred: Flexible, fast, and accurate object detection for autonomous driving systems. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20. Association for Computing Machinery, 2020. doi:10.1145/3341105.3373918.

39 Tesla Model S Owners Manual. https://www.tesla.com/sites/default/files/model_s_owners_manual_north_america_en_us.pdf, April 2020.

40 Vijay V. Vazirani. *Approximation Algorithms*. Springer Publishing Company, Incorporated, 2010. doi:10.1007/978-3-662-04565-7.

41 Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405, 2019. doi:10.1109/RTSS46320.2019.00042.

42 Shuochao Yao, Yifan Hao, Yiran Zhao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Tianshi Wang, Jinyang Li, and Tarek Abdelzaher. Scheduling real-time deep learning services as imprecise computations. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, 2020. doi:10.1109/RTCSA50079.2020.9203676.

43 Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving dataset for heterogeneous multitask learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. doi:10.1109/CVPR42600.2020.00271.

44 Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, Wen-Mei Hwu, and Deming Chen. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. In *Proceedings of Machine Learning and Systems 2020*, pages 216–229. 2020.

45 Husheng Zhou, Soroush Bateni, and Cong Liu. S^3DNN: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 190–201, 2018. doi:10.1109/RTAS.2018.00028.

46 Menglong Zhu and Mason Liu. Mobile video object detection with temporally-aware feature maps. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5686–5695, 2018. doi:10.1109/CVPR.2018.00596.

47 Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7218, 2018. doi:10.1109/CVPR.2018.00753.

48 Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 408–417, 2017. doi:10.1109/ICCV.2017.52.

49 Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4141–4150, 2017. doi:10.1109/CVPR.2017.441.