This WACV 2023 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

ROMA: Run-Time Object Detection To Maximize Real-Time Accuracy

JunKyu Lee
Queen's University Belfast
Belfast, UK
junkyu.lee@qub.ac.uk

Blesson Varghese
University of St Andrews
St Andrews, UK
blesson@st-andrews.ac.uk

Hans Vandierendonck
Queen's University Belfast
Belfast, UK
h.vandierendonck@qub.ac.uk

Abstract

This paper analyzes the effects of dynamically varying video contents and detection latency on the real-time detection accuracy of a detector and proposes a new run-time accuracy variation model, ROMA, based on the findings from the analysis. ROMA is designed to select an optimal detector out of a set of detectors in real time without label information to maximize real-time object detection accuracy. ROMA utilizing four YOLOv4 detectors on an NVIDIA Jetson Nano shows real-time accuracy improvements by 4 to 37% for a scenario of dynamically varying video contents and detection latency consisting of MOT17Det and MOT20Det datasets, compared to individual YOLOv4 detectors and two state-of-the-art runtime techniques.

1. Introduction

Real-time object detection plays a fundamental role in various applications such as self-driving cars, real-time object tracking, real-time activity recognition, and robotics [23, 13, 10]. However, using a single detector is limited in improving detection performance on dynamically varying video contents and dynamically varying compute resources, due to a fixed backbone network. In this regard, many run-time techniques explored how to select an optimal selector out of multiple detectors to improve the detection performance [13] or to save compute resources, given an accuracy budget [9, 14]. *E.g.*, switching between multiple detectors in real-time according to object size distribution in test data can improve the detection performance, compared to utilizing a single detector [13].

Previous approaches utilizing multiple detectors [13, 9, 14] are limited in practice for real-time object detection applications. *E.g.*, a run-time approach selects an optimal network based on periodic accuracy assessment, requiring labels from data [9]. However, the ground truths are not known in real-time for many real-time applications. Another run-time approach selects an optimal network assuming that available compute resources are fixed [13]. However, the available compute resources can vary in practice according to the background workload. Lou et al. [14] considered dynamically varying compute resources but did not consider the impact of dynamically varying object sizes and speeds on the accuracy. Therefore, a natural question arises: how can an optimal detector be selected without accessing labeled data based on the effects of both dynamically varying video contents and available compute resources? To the best of our knowledge, no solution to this problem exists in the literature.

Real-time accuracy highly depends on the available compute resources, unlike offline detection accuracy. *E.g.*, if a computing device is shared to analyze multiple video streams, adding a new video analysis task will reduce the amount of computation available to the other tasks. The reduced compute resources increase the detection latency, degrading the real-time performance. The increased latency can be addressed by dropping frames [13, 10] or downsampling frames [9, 12]. As such, the computational efficiency of real-time video analytics is tightly bound to its real-time accuracy.

We model the effects of dynamically varying object sizes, moving speeds, and detection latency on the real-time accuracy of each detector by separating the effects into two parts: the effects of dynamically varying objects on the accuracy and the effects of dynamically varying object speeds and latency on the accuracy. We establish a new model based on this idea and this model can be used to choose the best performing detector out of multiple detectors without label information. The model estimates are computed based on information available at run-time: characteristics of the objects detected in the current and previous frame, and the detection latency. We name our run-time accuracy estimation model as Runtime Object Detection Accuracy Variation Estimation to Maximize Real-Time Accuracy (ROMA). The main contributions of this paper include:

- Analysis of the effects of dynamically varying compute resources and video contents on the real-time accuracy variation.

- A novel run-time accuracy estimation model, ROMA, that estimates the Relative Average Precision (RAP) of each detector to a currently running detector without label information. ROMA is designed, independently of detector types and computing platforms.
- Demonstrating ROMA using multiple YOLOv4 detectors [3] on an NVIDIA Jetson Nano with the Multiple Object Tracking Challenge 2017 (MOT17Det) and 2020 (MOT20Det) datasets [1] for dynamically varying video content and compute resources use cases.

We present related work in section 2, analysis of real-time accuracy variation and the implementation of ROMA in section 3, the experimental evaluation in section 4, and conclude our paper in section 5.

2. Related Work

Several researchers attempted to control the frame rate of video streams at run-time according to dynamically varying video contents to improve real-time detection accuracy [9, 12, 10]. Korshunov et al. [10] discussed that slower-moving objects were correctly detected when a higher fraction of the frames were dropped, implying that high frame rates were unnecessary in such cases. Mohan et al. [18] estimated the least sufficient Frame Per Second (FPS) during run-time for object tracking applications to save bandwidth between cameras and the compute devices, given the accuracy budgets. *E.g*., if the tracked object is within a distance threshold on the subsequent frames, the FPS is lowered until it violates the threshold.

Zoph et al. [30] proposed Neural Architectural Search (NAS) to seek optimal DNN models in the space of hyperparameters of network width, depth, and resolution. Since then, many NAS variants attempted to seek resource-efficient DNNs to deploy them on resource-constrained devices [7, 20, 21, 22, 24, 2, 4]. *E.g*., Tan et al. [22] proposed a resource-aware NAS, *Efficientdet*, to seek resource-efficient detectors for object detection applications. Recently, a feed-forward NAS approach [4] produced resource-efficient DNNs, given computing resource and latency constraints.

A different strand of work aims to select the most appropriate detector from a set of available detectors [14, 13, 9] or an appropriate channel width out of a single multi-capacity detector [5, 28]. Lee et al. [13] exploit temporal locality and select an optimal detector in real-time based on the sizes of objects in the video. Boundaries between object sizes were empirically determined using the MOT17Det dataset [1]. Their run-time technique "Transprecise Object Detection (TOD)" selects the detector that corresponds to the median object size found in the last frame. Lou et al. [14] propose a Latency-Aware Detection (LAD) run-time technique that selects the detector with the highest latency that meets the required frame rate, which corresponds to the detector with the highest off-line detection accuracy that meets the frame rate. Both TOD [13] and LAD [14] required multiple resource-efficient detectors to be uploaded to DRAM at initialization time to minimize the time overhead of switching detectors. *E.g*., TOD required an 11% additional memory footprint to upload four different detectors on an NVIDIA Jetson Nano device, compared to uploading a single heavy-weight detector out of the four detectors [13].

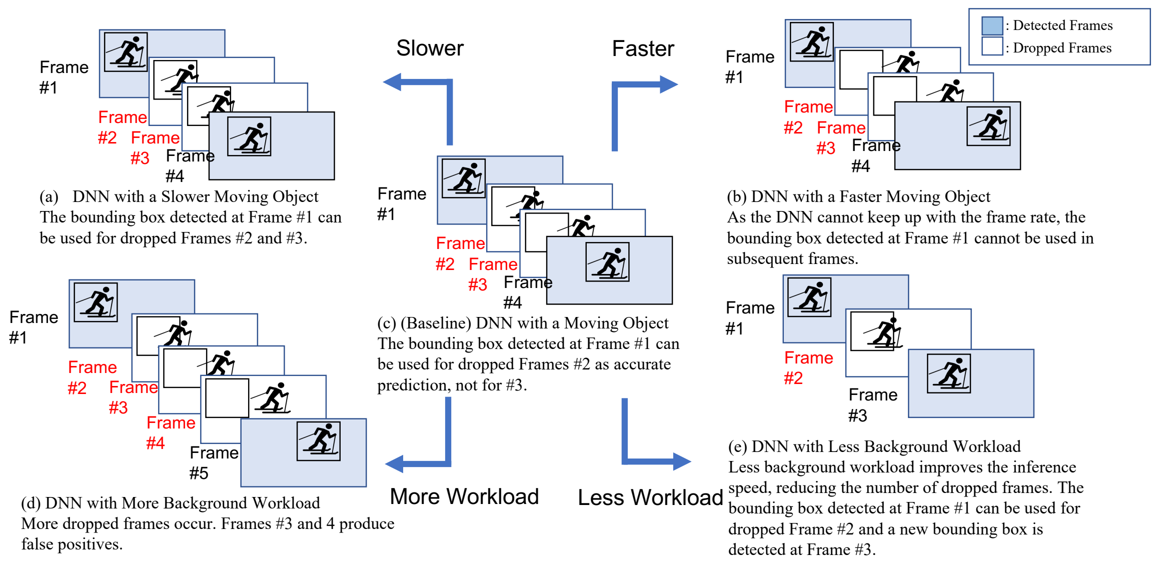
Yu et al. [29] propose the run-time technique of pruning unimportant neurons. A run-time decision maker [16] switched between multiple detectors during run-time to improve image classification accuracy. Fang et al. [5] implemented a multi-capacity DNN, "NestDNN", and dynamically selected an optimal sub-DNN of NestDNN to improve image classification accuracy given compute resources. Minhas et al. [17] selected an appropriate detector model according to dynamically varying accuracy constraints to improve the inference throughput.

A lightweight object tracking algorithm can be utilized to improve bounding boxes locations at dropped frames. For example, a Faster RCNN [19] coupled with various object tracking algorithms were used as a new detector candidate, and a decision maker chose the best configuration of a Faster RCNN (e.g., the number of region proposals and a DNN input resolution) coupled with an optimal object tracking algorithm according to dynamically varying available compute resources and object moving speeds [26, 27, 25]. The decision makers [26, 25] considered the average of object sizes, but the average of object sizes is limited in representing the histogram of object sizes appeared in a frame.

Each of these works addresses some aspect that ROMA solves. However, none of these works provides an optimal solution that handles all requirements at the same time: operating without labeled data, adapting to dynamically varying object sizes and speeds, and adapting decisions based on dynamically changing compute budget. It is possible to incorporate object tracking algorithm to infer bounding boxes locations at dropped frames. The benefits may be minimal due to the way mAP is measured. *E.g*., a minor correction on the position with a tracking algorithm would have limited impact on whether the overlap is higher or lower than 50%. Therefore, we did not pursue this. In addition, the latency of tracking (e.g., $20 - 550ms$ on an NVIDIA Jetson TX2 [27]) would drop many frames to maintain real-time processing, which would cause a reduction in accuracy. We leave it as an open question whether ROMA can be improved by incorporating a tracker.

3. ROMA: Run-Time Accuracy Variation

We discuss the real-time accuracy trade-off between the offline accuracy and the latency of a detector using the Average Precision (AP) metric. Later, we discuss the ROMA.



(a) DNN with a Slower Moving Object
The bounding box detected at Frame #1 can be used for dropped Frames #2 and #3.

(b) DNN with a Faster Moving Object
As the DNN cannot keep up with the frame rate, the bounding box detected at Frame #1 cannot be used in subsequent frames.

(c) (Baseline) DNN with a Moving Object
The bounding box detected at Frame #1 can be used for dropped Frames #2 as accurate prediction, not for #3.

(d) DNN with More Background Workload
More dropped frames occur. Frames #3 and 4 produce false positives.

(e) DNN with Less Background Workload
Less background workload improves the inference speed, reducing the number of dropped frames. The bounding box detected at Frame #1 can be used for dropped Frame #2 and a new bounding box is detected at Frame #3.

Figure 1. Accuracy Variation with Dynamically Varying Objects' Speeds and Available Compute Resources

3.1. Real-Time Accuracy Characteristics

Central to our approach to processing the video stream in real-time without running behind is to drop frames when the frame rate cannot be achieved [10, 13]. When frames are dropped, we apply the same object detection bounding boxes as the previous frame that was analyzed. Figure 1 analyzes the key problems that may occur when copying the detection bounding boxes from analyzed frames to subsequently dropped frames.

Consider case (c) (Figure 1, center) as a baseline. As the skier moves, it leaves the bounding box identified for frame #1. In frame #2, the overlap between the previously identified bounding box and the skier is sufficient to consider a correct detection (*e.g.*, an Intersection of Union (IoU) is larger than 50%). However, in frame #3, the skier has moved further and the detection fails, based on information in frame #1. As such, dropping frame #2 still allows obtaining a correct prediction, while dropping frame #3 results in an incorrect prediction.

Accuracy for dropped frames is dependent on the video contents, in particular the speed at which objects move and the original video frame rate. Case (a) shows a scenario where the movement of the skier is less than in case (c). In this case, the skier can be detected correctly in both frames #2 and #3 when those frames are dropped. In case (b), the speed of the skier is higher, and the bounding box found in frame #1 quickly becomes stale in both frames #2 and #3.

Another major factor that impacts on real-time object detection accuracy is computational latency. Computational latency may vary across object detectors (trading-off complexity of the detector against its accuracy), or when the compute hardware is shared with other workloads. When it takes longer to analyze a frame, then a higher number of subsequent frames need to be dropped in order to keep up with the real-time frame rate (case (d)). Alternatively, if frames can be analyzed more quickly (case (e)), a higher fraction of frames can be analyzed. As such, one expects a higher detection accuracy.

In real-time object detection, the choice of object detector has a complicated impact on accuracy, as it simultaneously impacts on several factors. A heavyweight detectorrequires more computation to be performed, while achieving higher accuracy than a lightweight detector [8]. However, by requiring more computation, a higher number of frames will need to be dropped (Figure 1, case (d)), which negatively impacts on accuracy. Similarly, a lightweight detector may be able to analyze more frames, but it does so with less accuracy than the heavyweight detector.

Additionally, the contents of the video frames impacts on accuracy. *E.g.*, lightweight detectors achieve comparable accuracy to heavyweight detectors on large objects, but not so on small objects [8]. The goal of this paper is to detangle the complex interaction between video content and latency of executing the object detector for real-time accuracy of an object detector. As a use case, we exemplify one of stage-of-the-art YOLOv4 detectors [3]. A YOLOv4 employs either a 9-layered DNN backbone for a tiny version [6] or a 53-layered DNN backbone for a full version [3]. The two knobs, the DNN's resolution r_{DNN} and the detector structure (i.e., tiny vs full version), can be used to control the trade-off between speed and offline AP of YOLOv4 in real-time. *E.g.*, smaller objects can be detected more by employ-

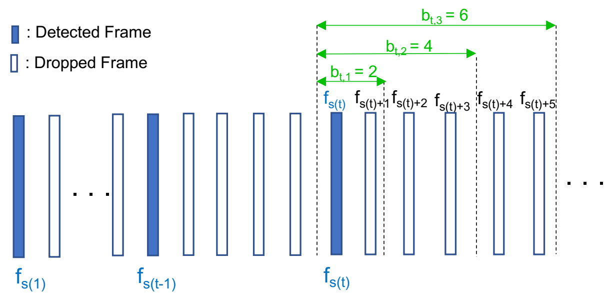


Figure 2. Notations for Frames and Frame Block Sizes

ing a higher r_{DNN} or a full version detector. However, the longer detection latency can hurt real-time accuracy.

While the ROMA model is informed by experiments with YOLOv4s, other detectors might have different characteristics not present in the YOLO detectors, and may require an extension of the model.

3.2. Notations for Frames and Frame Block Sizes

We describe our model that estimates RAPs in this section. In the beginning, we use the notations for n detector candidates in a detector pool $\mathbf{d}$ as follows:

$$\mathbf{d} = \{d_1, ..., d_n\}. \quad (1)$$

A video stream consists of a sequence of frames, which are analyzed one by one by an object detector. The i_f^{th} frame in the sequence is identified as f_{i_f}, where $i_f = 0, 1, 2, \ldots$. Some frames are analyzed by the detector, while others are dropped when frames arrive faster than they can be analyzed. The model estimates RAPs using a series of frames consisting of the detected frame and subsequently dropped frames. We refer to such a series of frames associated with a detector as *a frame block* of a detector and the number of frames consisting of a frame block as *a frame block size*. *E.g.*, the first frame in a frame block is analyzed by the object detector and the remaining ones have been dropped. Therefore, the first analyzed frame is commonly used for each frame block associated with each detector and the frame block size of each detector depends on the number of dropped frames of a detector. Thus, we use the notation $f_{s(t)}$ for the t^{th} analyzed frame, where $t = 0, 1, 2, \ldots$ and $s(t)$ associates with a corresponding frame index i_f. This way, we can link the analyzed frame index t to a frame sequence number i_f.

Fig. 2 describes our mathematical notations for frames and frame block sizes with an example for the number of detectors $n = 3$. As such, a frame block starting with $f_{s(t)}$ consists of frames $f_{s(t)}, f_{s(t)+1}, f_{s(t)+2}, \ldots, f_{s(t)+b_{t,i}-1}$, where $b_{t,i}$ indicates the *frame block size* of a frame block starting at $f_{s(t)}$ associated with a detector d_i. Likewise, we use the notation $d_{c(t)}$ for a currently chosen detector to detect objects at the frame $f_{s(t)}$, where $c(t)$ associates with a detector index i. We will estimate RAP of each detector d_i based on *its own frame block*, compared to a currently running detector $d_{c(t)}$.

The AP at $f_{s(t)}$ is the offline AP of a detector, and the AP is expected to degrade gradually as the number of dropped frames increase in proportion to the expected number of missing objects due to IoU deviation between $f_{s(t)}$ and a dropped frame. Hence, ROMA estimates the offline AP of each detector and then seeks the accuracy degradation rate at each dropped frame.

3.3. Offline AP Estimation of Each Detector

If the precision distribution is equivalent among the objects detected from each detector, the recall becomes the main factor in determining the AP. The recall is improved in proportion to the number of detected objects which highly depends on object size distribution on a video frame. Based on the above assumption, we estimate the number of objects detected of each detector d_i for offline AP estimation using the object size distribution detected at $f_{s(t)}$ using $d_{c(t)}$.

To do so, we first seek the detection performance ratios between d_i and $d_{c(t)}$ at different object size regions using an offline dataset (i.e., not used for evaluation dataset). We divide the object sizes (i.e., the number of pixels) into the H regions and measure the number of detected objects of each detector d_i to form an $\mathbf{p}_i$ vector as follows:

$$\mathbf{p}_i = [p_1(d_i), p_2(d_i), ..., p_H(d_i)]^T, \quad (2)$$

where $p_k(d_i)$ is the number of detected objects at the region k using the detector d_i. Utilizing each $\mathbf{p}_i$ generates a prior histogram matrix $\mathbf{P}$ as follows:

$$\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n]^T. \quad (3)$$

Next, the number of objects directly detected on $f_{s(t)}$ using the current detector $d_{c(t)}$ with respect to each region can generate a vector $\tilde{\mathbf{p}}_{t,c(t)}$ as follows:

$$\tilde{\mathbf{p}}_{t,c(t)} = [\tilde{p}_1(d_{c(t)}), \tilde{p}_2(d_{c(t)}), ..., \tilde{p}_H(d_{c(t)})], \quad (4)$$

where $\tilde{p}_k(d_{c(t)})$ is the number of objects detected at region k on $f_{s(t)}$ using the detector $d_{c(t)}$.

Next, the relative number of detected objects of d_i to $d_{c(t)}$ at each region is estimated as follows:

$$\mathbf{r}_{i,c(t)} = [p_1(d_i)/p_1(d_{c(t)}), ..., p_H(d_i)/p_H(d_{c(t)})]^T. \quad (5)$$

Notice that $\tilde{p}_k(d_{c(t)})$ is run-time detection information which varies over time according to video contents on $f_{s(t)}$ while $p_k(d_{c(t)})$ is offline detection information using offline data, which is fixed over time. From this point forward, we will use tilde notations for data measured during run-time (e.g., $\tilde{\mathbf{p}}_{t,c(t)}$).

Finally, we estimate the number of objects detected on $f_{s(t)}$ using a detector d_i by utilizing both the run-time information $\tilde{\mathbf{p}}_{t,c(t)}$ and the detection ratio information $\mathbf{r}_{i,c(t)}$:

$$l_{t,i} = \mathbf{r}^T_{i,c(t)}\tilde{\mathbf{p}}_{t,c(t)}. \quad (6)$$

If $i = c(t)$, we use the tilde notation, $\tilde{l}_{t,c(t)}$, since the number of detected objects at $f_{s(t)}$ is directly measured rather than estimated.

3.4. AP Degradation at Each Dropped Frame

We estimate the AP degradation rate at each j^{th} dropped frame (e.g., $f_{s(t)+j}$, where $j \geq 1$), compared to the AP on the analyzed frame $f_{s(t)}$. We use the notation $AP_{t,i}(f_{s(t)+j})$ for the estimated AP of a detector d_i at the frame $f_{s(t)+j}$ in a frame block and the notation $\overline{AP}_{t,i}$ for the estimated average AP of d_i over frames in a frame block starting with the frame $f_{s(t)}$:

$$\overline{AP}_{t,i} = \Sigma_{j=0}^{b_{t,i}-1} AP_{t,i}(f_{s(t)+j})/b_{t,i}. \quad (7)$$

Each $AP_{t,i}(f_{s(t)+j})$ is either lower than or equal to $AP_{t,i}(f_{s(t)})$, since some of detected objects on $f_{s(t)}$ can be lost due to limited overlap between objects' bounding boxes multiple frames apart. In this regard, we model $AP_{t,i}(f_{s(t)+j})$ by introducing an accuracy degradation ratio parameter as follows:

$$AP_{t,i}(f_{s(t)+j}) = AP_{t,i}(f_{s(t)}) \times \beta_{s(t)+j}, \quad (8)$$

where each $\beta_{s(t)+j}$ represents an AP degradation ratio at the frame $f_{s(t)+j}$, compared to the AP at $f_{s(t)}$ (e.g., $\beta_{s(t)} = 1$ always and $0 \leq \beta_{s(t)+j} \leq 1$, where $j \geq 1$). We notice that $\beta_{s(t)+j}$ can be shared among all detectors, since the ratio mainly relies on the average of detected objects' moving speeds depending on video contents rather than a detector type.

We estimate $\beta_{s(t)+j}$ with three steps. In step 1, we estimate the frame block size $b_{t,i}$ of d_i. In step 2, we estimate the number of missing objects per dropped frame due to the IoU deviations using $b_{t,i}$. In step 3, $\beta_{s(t)+j}$ is estimated using the estimated number of missing objects per frame.

For the step 1, the detection latency of d_i, $L_{t,i}$, determines $b_{t,i}$. If a detector is not switched between $f_{s(t-1)}$ and $f_{s(t)}$, $L_{t,i}$ is updated as follows:

$$L_{t,i} = (\tilde{L}_{t,c(t)}/L_{t-1,c(t)}) \times L_{t-1,i}, \text{ if } c(t) = c(t-1). \quad (9)$$

This way, if available compute resources varies between $f_{s(t-1)}$ and $f_{s(t)}$, $L_{t,i}$ is updated by using a latency variation ratio $\tilde{L}_{t,c(t)}/L_{t-1,c(t)}$. If the detector is changed between $f_{s(t-1)}$ and $f_{s(t)}$ (i.e., $c(t-1) \neq c(t)$), the estimated latency $L_{t,c(t)}$ is directly updated to the measured latency $\tilde{L}_{t,c(t)}$:

$$L_{t,c(t)} = \tilde{L}_{t,c(t)}, \text{ if } i = c(t) \text{ and } c(t) \neq c(t-1). \quad (10)$$

The rest of estimated latency $L_{t,i}$ of the other detectors are unchanged if $c(t-1) \neq c(t)$:

$$L_{t,i} = L_{t-1,i} \text{ if } i \neq c(t) \text{ and } c(t) \neq c(t-1). \quad (11)$$

We do not utilize Eq. (9) for $c(t) \neq c(t-1)$, so that the update of other detectors' latency utilizes the latency ratio derived only from the direct measurements (i.e., $L_{t-1,c(t)} = \tilde{L}_{t-1,c(t)}$ if $c(t) = c(t-1)$). Using an $L_{t,i}$ and an FPS constraint FPS, the $b_{t,i}$ is estimated as follows:

$$b_{t,i} = f(FPS, L_{t,i}) = \lfloor FPS \times L_{t,i} \rfloor + 1. \quad (12)$$

Notice that each $b_{t,i}$ is varying with t according to the availability of compute resources.

For the step 2, we estimate the number of missing objects per frame due to IoU deviation between bounding boxes detected at $f_{s(t-1)}$ and $f_{s(t)}$. To do so, we measure the number of objects $\tilde{m}_t$ during run-time that satisfy an IoU threshold between the two consequent detected frames based on Algorithm 1. Using $\tilde{m}_t$, we seek the number of objects,

Algorithm 1 Measuring the number of objects satisfying an IoU threshold

```
$\tilde{m}_t = 0$  // Initialize the number of survived objects.
for $i_o$ = 1, 2, ..., $\tilde{l}_{(t-1),c(t-1)}$ do
    for $j_o$ = 1, 2, ..., $\tilde{l}_{t,c(t)}$ do
        Measure IoU[$i_o$][$j_o$] between the two bounding boxes of $i_o$ and $j_o$
        if IoU[$i_o$][$j_o$] $\geq$ IoU threshold then
            $\tilde{m}_t = \tilde{m}_t + 1$
            break
        end if
    end for
end for
```

$\bar{m}_t$, that violates an IoU threshold between the two detected frames as follows:

$$\bar{m}_t = \tilde{l}_{(t-1),c(t-1)} - \tilde{m}_t, \quad (13)$$

where $\tilde{l}_{(t-1),c(t-1)}$ is the number of detected objects at $f_{s(t-1)}$ measured by $d_{c(t-1)}$. Now, we can estimate the number of missing objects per frame, u_t, as follows:

$$u_t = \bar{m}_t/b_{t,c(t)}. \quad (14)$$

For the step 3, we estimate the number of objects detected at $f_{s(t)+j}$, $q_{s(t)+j}$, using u_t iteratively as follows:

$$q_{s(t)+j} = q_{s(t)+j-1} - u_t, \quad (15)$$

where $q_{s(t)} = \tilde{l}_{t,c(t)}$. Notice that we leverage temporal locality and assume that u_t (measured between $f_{s(t-1)}$ and $f_{s(t)}$) can be applied for the frames from

$f_{s(t)}$ to $f_{s(t+1)}$. The u_t objects out of $q_{s(t)+j-1}$ objects at $f_{s(t)+j-1}$ are generally switched from TPs to FPs at $f_{s(t)+j}$, letting both precision and recall drop in proportion to the ratio of $(q_{s(t)+j}/q_{s(t)+j-1})$ at $f_{s(t)+j}$, compared to $f_{s(t)+j-1}$. Therefore, it is highly probable that the AP at the frame $f_{s(t)+j}$ drops quadradically in proportion to $(q_{s(t)+j}/q_{s(t)+j-1})$, compared to the frame $f_{s(t)+j-1}$. Now, we estimate $\beta_{s(t)+j}$ based on this observation:

$$\beta_{s(t)+j} = \beta_{s(t)+j-1} \times (q_{s(t)+j}/q_{s(t)+j-1})^2. \quad (16)$$

3.5. Estimating Relative Average Precision

The RAP of d_i to $d_{c(t)}$, $a_{t,i}$, can be expressed using an offline accuracy ratio between the two detectors, $\alpha_{t,i}$, and an accuracy degradation ratio between the two detectors, $\gamma_{t,i}$, as follows:

$$a_{t,i} = \overline{AP}_{t,i}/\overline{AP}_{t,c(t)} = \alpha_{t,i} \times \gamma_{t,i}, \quad (17)$$

where

$$\alpha_{t,i} = AP_{t,i}(f_{s(t)})/AP_{t,c(t)}(f_{s(t)}) \approx l_{t,i}/\tilde{l}_{t,c(t)} \quad (18)$$

and

$$\gamma_{t,i} = (\Sigma_{j=0}^{b_{t,i}-1}\beta_{s(t)+j}/b_{t,i})/(\Sigma_{j=0}^{b_{t,c(t)}-1}\beta_{s(t)+j}/b_{t,c(t)}). \quad (19)$$

ROMA chooses one of d_is that has the index i of the maximum $a_{t,i}$.

3.6. Implementation of ROMA

This section exemplifies the implementation of ROMA (i.e., Eq. (17)) in terms of the initialization process and running process.

3.6.1 Initialization

At initialization time, multiple detectors are uploaded to DRAM. An FPS constraint, FPS, is found based on a video file. The initial frame block size $b_{i,0}$ uses the prior latency information of the detector d_i on a compute platform. The histogram matrix $\mathbf{P}$ is found using a video dataset unseen from the evaluation dataset. The default detector is chosen as the slowest detector. The maximum frame block size is set to 30, and the all $\beta_{s(0)+j}$s are initialized to '1' for $0 \leq j \leq 29$.

3.6.2 Running Process

For the updates of $\alpha_{t,i}$ in Eq (18), the $l_{t,i}$s are estimated using Eq. (6). To prevent the division by zero in Eq. (18), we add 0.1 to the divisor. The $\mathbf{r}_{t,i}$ is found using Eq. (5) and the $\tilde{\mathbf{p}}_{t,c(t)}$ in Eq. (6) is found using the detected bounding boxes information at the frame $f_{s(t)}$ using $d_{c(t)}$. Each frame block size $b_{t,i}$ is updated based on Eq. (12). Depending on whether $d_{c(t)}$ is changed between $f_{s(t-1)}$ and $f_{s(t)}$, each $L_{t,i}$ is estimated using Eq. (9) for $c(t) = c(t-1)$ or Eq. (10) and Eq. (11) for $c(t) \neq c(t-1)$. The number of objects satisfying an IoU threshold is computed based on Algorithm 1 and the number of missing objects per frame u_t is computed using Eq. (14). Each $\beta_{s(t)+j}$ is computed based on Eq. (15) and Eq. (16). The detector with the index i that has the maximum value of $a_{t,i}$ in Eq. (17) is selected to be run at the frame $f_{s(t+1)}$.

If $b_{t,c(t)} < b_{t,i}$, no run-time information is available for the $\beta_{s(t)+j}$ updates where $j \geq b_{t,c(t)}$. In this case, we leverage the ratio of $\beta_{s(t-1)+j}/\beta_{s(t-1)+j-1}$ for the $\beta_{s(t)+j}$ updates as follows:

$$\beta_{s(t)+j} = \beta_{s(t)+j-1} \times \beta_{s(t-1)+j}/\beta_{s(t-1)+j-1}. \quad (20)$$

Eq. (20) can update $\beta_{s(t)+j}$ up to the maximum frame block size depending on the detectors d_is. We address another special case in which the accuracy of $\beta_{s(t)+j}$ can suffer from noise of bounding boxes severely when $b_{t,c(t)}$ is low. To mitigate its effect, we set up a minimum frame block size threshold to update $\beta_{s(t)+j}$: $b_{th} = 3$. *E.g.*, if a frame block size of a current detector is larger than or equal to b_{th}, we update $\beta_{s(t)+j}$. Otherwise, we utilize $\beta_{s(t-1)+j}$s for $\beta_{s(t)+j}$s.

4. Experimental Evaluation

The experimental setting is as follows:
- Computing Platform: An NVIDIA Jetson Nano Board (MAX power mode).
- Object Detectors: YOLOv4-Tiny-288 (YT288), YOLOv4-Tiny-416 (YT416), YOLOv4-Full-288 (YF288), and YOLOv4-Full-416 (YF416) optimized by TensorRT with an FP16 (i.e., half precision) option. The confidence score thresholds are set to 0.3 for all YOLOs. The IoU threshold in Algorithm 1 is set to 0.5.
- Evaluation Datasets: MOT17Det and MOT20Det [1].
- Prior Histogram Matrix $\mathbf{P}$: $\mathbf{P}$ with $H = 3$, generated using four MOT15 datasets named ETH-Bahnhof, ETH-Sunnyday, TUD-Campus, and TUD-Stadtmitte, each having 640×480 resolutions [11].

$$\mathbf{P} = \begin{bmatrix} 1921 & 3550 & 2748 \\ 4603 & 3872 & 2488 \\ 8502 & 3506 & 2982 \\ 9526 & 3603 & 2993 \end{bmatrix} \quad (21)$$

We chose $H = 3$, each for small object size region r_s, medium object size region r_m, and large object size region r_l. We set up the object size boundaries for $s_1 = 2500$ $(pixel^2)$ between r_s and r_m, and $s_2 = 7500$ between r_m and r_l with respect to a 640×480 resolution video frame so that each region can contain at least 20% of detected objects

out of total detected objects across the three regions.
- Comparison with State-Of-The-Art Techniques: YT288, YT416, YF288, YF416, TOD [13], and LAD [14]. Notice that LAD in our paper utilizes the four YOLOv4 models instead of the detectors generated from [4]. It downgrades a detector to the next lighter detector (e.g., YT416 to YT288) when the latency violates an FPS constraint and upgrades a detector to the next heavier detector (e.g., YT288 to YT416) when the inference latency is lower than 30% of the latency constraint.
- Accuracy Evaluation Tool: MATLAB interface MOT evaluation tool kit provided by [1] (i.e., an 11-point interpolation assessment using an IoU threshold of 0.5). If the precision is reported as '0' at the recall point '0' based on the evaluation tool, we take a precision at the recall point as: $p(r) = \max_{r'} p(r')$, where $p(r)$ is the precision value at a recall point r and $r' \geq r$ [15].
- Real-Time AP: We measure the real-time object detection accuracy as used in [13, 9]; the bounding box information detected from the previous frame was used for the AP assessment for the subsequent dropped frames.

4.1. Real-Time AP Measurements

We evaluate real-time APs for TOD, LAD, the four different YOLOv4s, and ROMA on MOT17/20Det datasets while imposing four different workloads as shown in Table 1: case (a) for no background workload, case (b) for background workload with running a YT288, case (c) for background workload with running a YT416, and case (d) for background workload with running a YF416.

Using MOT17Det, ROMA outperforms all single detectors and other run-time techniques in terms of the average APs for each case of (a) to (d) as shown with bold marks in Table 1. Table 1 shows that deploying one single resource-efficient detector limits the real-time accuracy for dynamically varying video contents and compute resources using MOT17Det (e.g., motivation of run-time techniques such as [9, 14, 13]). *E.g.*, deploying YF416 can be a good choice for case (a), but can be the worst choice for case (d). Notice that state-of-the-art detectors, posted on the MOT website [1], are very slow therefore very low real-time AP on embedded devices like the Jetson Nano.

Fig. 3 shows the average APs of all detectors across cases (a) to (d) using MOT17Det and MOT20Det, respectively. This scenario mimics a dynamically varying compute resources and video content scenario in which each of the four different compute resources (i.e., (a) to (d)) is available for $1/4$ of the entire execution time and each video dataset is included in proportion to the number of frames of the dataset. The ROMA shows the accuracy improvements of 1.23, 1.03, 1.05, 1.10, 1.28, and 1.06 $\times$ compared to YT288, YT416, YF288, YF416, LAD, and TOD, respectively using MOT17Det, even though there are effectively

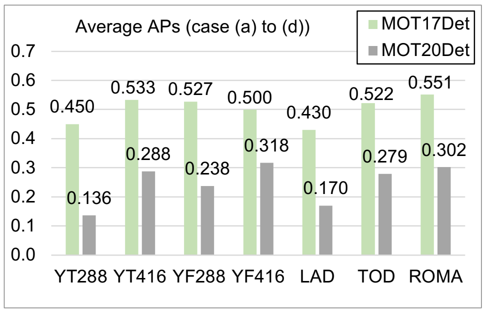


Figure 3. Average APs (MOT17Det and MOT20Det)

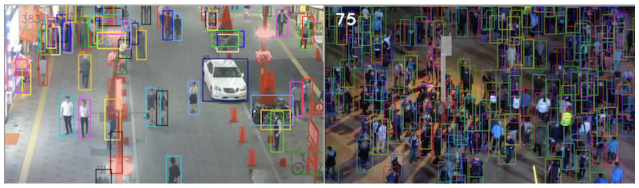


Figure 4. MOT17-04 (Left) and MOT20-05 (Right) [1]

fewer valuable YOLOs to choose from. Therefore, the AP difference between ROMA and any individual YOLO can be limited. This implies that ROMA is suitable for dynamically varying compute resources for each different video content case.

For MOT20Det, ROMA is the second best detector, following YF416, since MOT20Det contains more people than MOT17Det. *E.g.*, Fig. 4 shows MOT17-04 and MOT20-05, respectively. The time overhead of ROMA quadratically increases in proportion to the number of detected objects as shown in Algorithm 1. *E.g.*, the time overhead of ROMA on an NVIDIA Jetson Nano is measured as $6ms$ for case (a) on MOT17-04 and $12ms$ on MOT20-05. Notice that the time overhead does not depend on the number of objects on a video frame but on the number of detected objects using a detector. YF416 is chosen by ROMA with 100% for both MOT17-04 and MOT20-05 as shown in Fig. 7. Considering the detection latency of YF416 ($225ms$), the time overhead of ROMA did not affect the real-time accuracy on MOT17-04, but on MOT20-05 across case (a) to (d) based on Table 1. However, ROMA has equivalent performance to YF416 on the other MOT20Det datasets. Even though TOD and LAD have lower time overhead on average than ROMA (*e.g.*, $0.5ms$ for TOD and $0.01ms$ for LAD for MOT17Det), the decision of ROMA is more accurate than TOD and LAD, resulting in higher real-time accuracy.

Finally, we consider another scenario containing both MOT17Det and MOT20Det to compute the average AP of each detector across the four cases. In this scenario, Fig. 5 shows that ROMA is the best performing detector, showing 1.37, 1.04, 1.09, 1.06, 1.37, and $1.06\times$ performance improvement, compared to YT288, YT416, YF288, YF416, LAD, and TOD, respectively.

Table 1. Real-Time APs on MOT17/20Det with Background Workloads

Dataset	(a) With No Background Workload							(b) With YT288 Running							(c) With YT416 Running							(d) With YF416 Running						
	YT288	YT416	YF288	YF416	LAD	TOD	ROMA	YT288	YT416	YF288	YF416	LAD	TOD	ROMA	YT288	YT416	YF288	YF416	LAD	TOD	ROMA	YT288	YT416	YF288	YF416	LAD	TOD	ROMA
MOT17-02	0.35	0.36	0.44	0.50	0.35	0.51	0.50	0.35	0.36	0.43	0.48	0.34	0.49	0.48	0.35	0.36	0.43	0.47	0.35	0.48	0.47	0.35	0.36	0.41	0.34	0.32	0.36	0.41
MOT17-04	0.25	0.52	0.50	0.58	0.25	0.58	0.59	0.25	0.52	0.49	0.57	0.25	0.56	0.57	0.25	0.52	0.49	0.57	0.25	0.57	0.57	0.25	0.52	0.41	0.48	0.21	0.47	0.48
MOT17-05	0.78	0.79	0.79	0.77	0.79	0.78	0.79	0.78	0.79	0.78	0.74	0.79	0.78	0.79	0.78	0.79	0.77	0.73	0.79	0.78	0.79	0.78	0.79	0.76	0.63	0.68	0.76	0.77
MOT17-09	0.72	0.80	0.80	0.77	0.72	0.80	0.79	0.72	0.80	0.78	0.65	0.69	0.78	0.76	0.72	0.80	0.78	0.64	0.72	0.78	0.76	0.72	0.80	0.67	0.51	0.57	0.56	0.80
MOT17-10	0.25	0.34	0.41	0.44	0.25	0.43	0.41	0.25	0.34	0.39	0.35	0.25	0.35	0.38	0.25	0.34	0.39	0.35	0.25	0.35	0.38	0.25	0.34	0.36	0.31	0.12	0.21	0.36
MOT17-11	0.71	0.72	0.71	0.68	0.70	0.71	0.70	0.71	0.72	0.70	0.64	0.70	0.70	0.70	0.71	0.72	0.70	0.64	0.70	0.70	0.70	0.71	0.72	0.68	0.53	0.51	0.49	0.70
MOT17-13	0.09	0.25	0.20	0.18	0.15	0.18	0.21	0.09	0.23	0.18	0.16	0.15	0.17	0.19	0.09	0.17	0.18	0.16	0.09	0.17	0.19	0.09	0.16	0.13	0.12	0.09	0.11	0.19
AVG.	0.450	0.540	0.550	0.560	0.459	**0.570**	**0.570**	0.450	0.537	0.536	0.513	0.453	0.547	**0.553**	0.450	0.529	0.534	0.509	0.450	0.547	**0.551**	0.450	0.527	0.489	0.417	0.357	0.423	**0.530**
MOT20-01	0.26	0.44	0.35	0.45	0.26	0.36	0.45	0.18	0.44	0.35	0.44	0.26	0.35	0.44	0.15	0.44	0.35	0.44	0.26	0.35	0.43	0.14	0.44	0.35	0.31	0.22	0.31	0.42
MOT20-02	0.26	0.44	0.35	0.44	0.26	0.35	0.44	0.18	0.44	0.35	0.44	0.26	0.35	0.43	0.15	0.44	0.35	0.43	0.26	0.35	0.43	0.14	0.44	0.35	0.42	0.26	0.35	0.35
MOT20-03	0.09	0.18	0.17	0.26	0.09	0.26	0.26	0.09	0.18	0.17	0.26	0.09	0.26	0.26	0.09	0.18	0.17	0.26	0.09	0.26	0.26	0.09	0.18	0.17	0.25	0.09	0.24	0.25
MOT20-05	0.09	0.09	0.08	0.17	0.08	0.17	0.17	0.09	0.09	0.08	0.17	0.08	0.17	0.08	0.09	0.09	0.08	0.17	0.08	0.17	0.08	0.09	0.09	0.08	0.17	0.08	0.17	0.08
AVG.	0.175	0.288	0.238	**0.330**	0.173	0.285	**0.330**	0.135	0.288	0.238	**0.328**	0.173	0.283	0.303	0.120	0.288	0.238	**0.325**	0.173	0.283	0.300	0.115	0.288	0.238	**0.288**	0.163	0.268	0.275
TOT AVG.	0.350	0.448	0.436	0.476	0.355	0.466	**0.483**	0.335	0.446	0.427	0.445	0.351	0.451	**0.462**	0.330	0.441	0.426	0.442	0.349	0.451	**0.460**	0.328	**0.440**	0.397	0.370	0.286	0.366	0.437

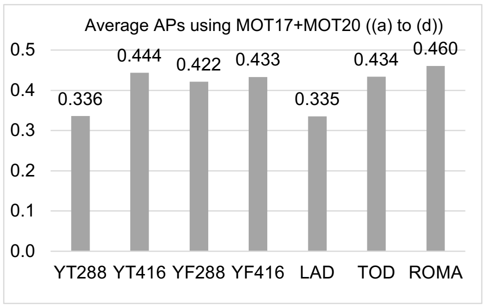


Figure 5. Average APs across All Cases (MOT17Det+MOT20Det)

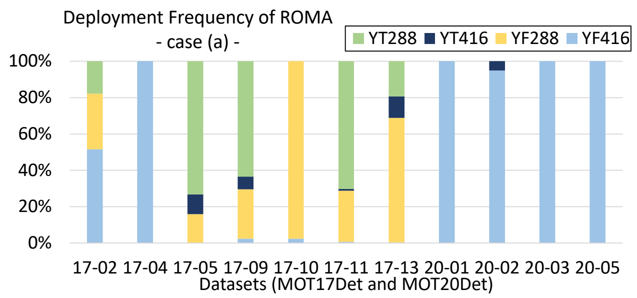


Figure 7. Deployment Frequency by ROMA

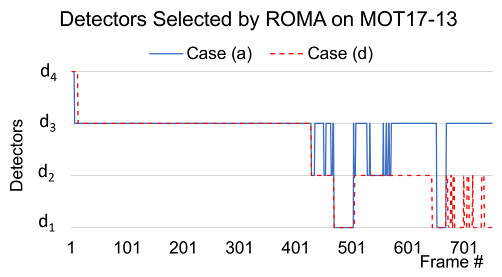


Figure 6. Detectors selected by ROMA on MOT17-13

4.2. Decisions by ROMA

Fig. 6 shows the decisions made by ROMA with $d_1 = YT288$, $d_2 = YT416$, $d_3 = YF288$, and $d_4 = YF416$ for case (a) and (d) on MOT17-13.ROMA downgrades the detectors used for case (a) to lighter detectors for case (d). A bus moves straight in the direction aligned with the camera in early frames and turns right later, increasing relative object speeds. ROMA switches a current detector to a lighter detector when objects move faster. With a low dynamic range of object sizes and the object moving speeds, the decision of ROMA was biased in early frames.

Fig. 7 shows the deployment frequency of each detector by ROMA on MOT17/20Dets. ROMA selects YF416 solely for MOT17-04, MOT20-01, 03, and 05 (*e.g.*, video frames captured by static cameras), while selects multiple detectors dynamically for MOT17-05, 09, 11, and 13 (*e.g.*, video frames captured by moving cameras). ROMA selects multiple detectors in MOT17-02 (static camera), since people walk in early frames and later kids riding bicycles appear, increasing relative object speeds.

5. Conclusion

Deploying a single object detector limits real-time accuracy on dynamically varying video contents and compute resources due to the fixed structure of the detector, which is the motivation of our paper. ROMA is designed to switch between multiple detectors without label information according to both dynamically varying video contents and available compute resources. This paper claims that the run-time information including the object size histograms, the IoUs approximation, and the detection latency is sufficient to estimate relative APs accurately for all detector candidates according to dynamically varying video contents and compute resources. ROMA demonstrates the best real-time accuracy, compared to individual detectors and two state-of-the-art run-time techniques.

Acknowledgment

This work is supported by the Engineering and Physical Sciences Research Council under grant agreement EP/T022345/1 and by CHIST-ERA under grant agreement CHIST-ERA-18-SDCDN-002 (DiPET).

References

[1] Multiple object tracking benchmark. `https://motchallenge.net`. Accessed: 29-August-2022.

[2] Andrew Anderson, Jing Su, Rozenn Dahyot, and David Gregg. Performance-oriented neural architecture search, 2020.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

[5] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. MobiCom '18, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Wei Fang, Lin Wang, and Peiming Ren. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, 8:1935–1944, 2020.

[7] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[8] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3296–3297, 2017.

[9] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, page 253–266, 2018.

[10] Pavel Korshunov and Wei Tsang Ooi. Reducing frame rate for object tracking. In *Proceedings of the 16th International Conference on Advances in Multimedia Modeling*, MMM'10, page 454–464, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking, 2015.

[12] Jeonghun Lee and Kwang il Hwang. Yolo with adaptive frame control for real-time object detection applications. *Multimedia Tools and Applications (2021)*, 2021.

[13] JunKyu Lee, Blesson Varghese, Roger Woods, and Hans Vandierendonck. TOD: Transprecise Object Detection to Maximise Real-Time Accuracy on the Edge. In *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, pages 53–60, 2021.

[14] Wei Lou, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V. Merrett. Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3104–3112, 2021.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[16] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Trans. Embed. Comput. Syst.*, 19(1), 2020.

[17] Umar Ibrahim Minhas, JunKyu Lee, Lev Mukhanov, Georgios Karakonstantis, Hans Vandierendonck, and Roger Woods. Increased leverage of transprecision computing for machine vision applications at the edge. *Journal of Signal Processing Systems*, 2022.

[18] A. Mohan, A. S. Kaseb, K. W. Gauen, Y. Lu, A. R. Reibman, and T. J. Hacker. Determining the necessary frame rate of video data for object tracking under accuracy constraints. In *IEEE Conference on Multimedia Information Processing and Retrieval*, pages 368–371, 2018.

[19] 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):1137–1149, 2017.

[20] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[21] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[22] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[23] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 107–122, Cham, 2020. Springer International Publishing.

[24] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[25] Ran Xu, Jayoung Lee, Pengcheng Wang, Saurabh Bagchi, Yin Li, and Somali Chaterji. Litereconfig: Cost and content aware reconfiguration of video object detection systems for mobile gpus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 334–351, New York, NY, USA, 2022. Association for Computing Machinery.

[26] Ran Xu, Fangzhou Mu, Jayoung Lee, Preeti Mukherjee, Somali Chaterji, Saurabh Bagchi, and Yin Li. Smartadapt: Multi-branch object detection framework for videos on mobiles. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2518–2528, 2022.

[27] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. Approxdet: Content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, page 449–462, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR '19: International Conference on Learning Representations*, 2019.

[29] R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[30] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *ICLR '17: International Conference on Learning Representations*, 2017.