2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00107

Exploiting Frame Similarity for Efficient Inference on Edge Devices

Ziyu Ying, Shulin Zhao*, Haibo Zhang*, Cyan Subhra Mishra, Sandeepa Bhuyan,
Mahmut T. Kandemir, Anand Sivasubramaniam, Chita R. Das
Department of Computer Science and Engineering, The Pennsylvania State University, USA
Email: {zjy5087, suz53, huz123, cyan, sxb392, mtk2, axs53, cxd12}@psu.edu

***Abstract*—Deep neural networks (DNNs) are being widely used in various computer vision tasks as they can achieve very high accuracy. However, the large number of parameters employed in DNNs can result in long inference times for vision tasks, thus making it even more challenging to deploy them in the compute- and memory-constrained mobile/edge devices. To boost the inference of DNNs, some existing works employ compression (model pruning or quantization) or enhanced hardware. However, most prior works focus on improving model structure and implementing custom accelerators. As opposed to the prior work, in this paper, we target the video data that are processed by edge devices, and study the similarity between frames. Based on that, we propose two runtime approaches to boost the performance of the inference process, while achieving high accuracy.**

Specifically, considering the similarities between successive video frames, we propose a frame-level compute reuse algorithm based on the motion vectors of each frame. With frame-level reuse, we are able to skip 53% of frames in inference with negligible overhead and remain within less than 1% mAP (accuracy) drop for the object detection task. Additionally, we implement a partial inference scheme to enable region/tile-level reuse. Our experiments on a representative mobile device (Pixel 3 Phone) show that the proposed partial inference scheme achieves $2\times$ speedup over the baseline approach that performs full inference on every frame. We integrate these two data reuse algorithms to accelerate the neural network inference and improve its energy efficiency. More specifically, for each frame in the video, we can dynamically select between (i) performing a full inference, (ii) performing a partial inference, or (iii) skipping the inference altogether. Our experimental evaluations using six different videos reveal that the proposed schemes are up to 80% (56% on average) energy efficient and $2.2\times$ performance efficient compared to the conventional scheme, which performs full inference, while losing less than 2% accuracy. Additionally, the experimental analysis indicates that our approach outperforms the state-of-the-art work with respect to accuracy and/or performance/energy savings.

***Index Terms*—Mobile Computing, DNN Inference, Energy Efficient, Video Analysis, Motion Vector, Object Detection**

I. Introduction

While video processing has become extremely popular on mobile devices, the next wave of emerging applications are likely to be those that analyze videos in a *faster* and *more efficient* fashion, to provide sophisticated intelligence. Such analytics are critical for virtual/augmented reality, object recognition/detection for surveillance, commercial advertisement insertion/deletion, synopsis extraction, and video querying. To enable such analytics, current devices rely heavily on deep neural networks (DNNs). However, DNNs are quite compute-intensive and generate very large memory footprints [1]. Furthermore, running inference for videos on edge devices is even more expensive than images due to the data volume and the power/energy constraints.

There has been recent research in optimizing video analytics on edge devices, targeting smartphones, autonomous driving cars, and VR/AR headsets. These optimizations have been attempted at different levels, including model compression (pruning [2], [3], quantization [4]), compiler support [2], [5]–[7], runtime systems [7]–[13], and hardware enhancements [9], [14], [15]. Let us now summarize the pros and cons of four representative state-of-the-art works which include various levels of optimizations as aforementioned, and compare them with our work. DeepCache [8] has been proposed for trading layer-wise intermediate data memoization/caching for computation savings, but it ignores the frame-wise reuse opportunities. From the hardware side, Euphrates [9] targets the frame-wise reuse, and customizes an accelerator for searching the regions of interest. Different from these two prior works, Potluck [12] caches the feature vectors (FVs) and corresponding inference results for key frames, and reuse the results for other frames with similar FVs, with the costs of FV extraction (e.g., down-sampling [16]) penalty and potential accuracy and/or performance drop due to sampling failures. On the other hand, MCDNN [7] proposes a scheduler to dynamically choose the best suitable model from several candidate models (e.g., standard model, medium-pruned variant, and heavily-pruned variant), and pushes the accuracy as high as possible within the energy budget. However, with limited energy budget on typical edge devices, the accuracy is far from sufficient for vision applications (quantitative results in Sec. V).

Although the prior works do add value to video analytics, each comes with its own problems and costs. In comparing these prior approaches with ours, we consider five critical features of DNN-based video optimization, shown in Table I: high accuracy, high performance improvement and energy savings, the hardware enhancement needed, generality of decision making logic for proper approximation and correspondingly, the adaptation to various runtime conditions. Specifically, with limited energy budget or available models, MCDNN suffers from accuracy drops. Also, if the approximation opportunity incurs high overhead and/or the decision is made based on high-level features, the scope for performance/energy savings

*Work was done while at Penn State.

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00107

TABLE I: Qualitative summary of DNN vision optimization work in terms of accuracy, saving, adaptivity, hardware support, and decision making mechanism (DM). Note that a check-mark in the second and third columns means the corresponding scheme achieves high accuracy and energy/performance efficiency, respectively. A check-mark in the Custom HW column indicates that the corresponding approach does not need custom hardware, whereas a cross means it needs.

	Accuracy	Saving	Adaptivity	Custom HW	DM
MCDNN [7]	✗	✓	✓	✓	Resource Budget
Potluck [12]	✓	✗	✓	✓	Feature Vector
Euphrates [9]	✓	✓	✗	✗	Reuse Distance
DeepCache [8]	✓	✗	✗	✓	Reuse Distance
This Work	✓	✓	✓	✓	Motion Vector

can be relatively low (as in Potluck [12] and DeepCache [8]). Additionally, sometimes sudden changes in subsequent frames (e.g., with a new object in the frame), require adaptive decisions, which cannot be achieved by Euphrates [9] and DeepCache [8]. Also, as opposed to Euphrates, one may prefer to use existing hardware, due to considerable development effort and associated cost with customization. Unlike the work presented in this paper, none of the four prior schemes mentioned in Table I performs well in all features listed in the table.

While one may hypothesize that a new software-hardware co-design may be a panacea for numerous applications, video analytics at the edge is not necessarily in this category. Edge devices are often equipped with mature and robust video and vision pipelines, including sophisticated hardware like the codecs, which the current video analytics pipeline is yet to fully exploit. Further, the video stream itself has temporal continuity, giving us the opportunities of memoization and reuse. In special cases, like remote surveillance, most frames can even be identical, avoiding the need to process them. Consequently, the only regions of interest (RoIs) should be the change between successive frames. Furthermore, these changes between consecutive frames need not be explicitly calculated as they can be extracted from the hardware codecs directly, giving us a chance to capitalize on an existing fundamental component, instead of adding new hardware blindly.

Leveraging inter-frame similarity, intra-frame redundancies, and the RoIs have been investigated for different purposes, e.g., exploring pixel-similarity to encode frames and reduce bandwidth consumption [17], utilizing RoIs to reduce the computational footprint [18], [19], analyzing inter-frame similarity to skip inferences [9], and caching the intermediate results to avoid redundant computation [8]. However, the granularity of similarity explored in these prior works is *static*, and either too coarse (e.g., skips the entire frame [9]) or too strict (e.g., still needs to compute despite very few changes). Hence, properly exploiting these concepts together in an "integrated" fashion for better optimizing neural inferencing is very much in its infancy that this paper proposes to address. Specifically, we present a systematic study of integrating the *temporal correlations* and *region-based inference* in a *unified* approach and try answering the following critical questions: *(i) what is the scope to explore both the pixel- and computational-similarities, i.e., frame-level, region-level or pixel-level?*, *(ii) can we exploit the temporal continuity of the video steam and safely skip the inference computation for similar frames?*, and if not, *(iii) can we just focus on the RoIs in the current frame and significantly reduce the computational requirements for its inferencing?*

Towards this, we propose and thoroughly evaluate a new approach to answer the aforementioned questions. Specifically, our approach combines inter-frame similarities, motion vectors (MVs), and the concept of regions of interest, to make online video analytics/inferences inherently faster, along with the runtime support for improving the video streaming pipeline. Our main **contributions** in this work include the following:

- First, we study the similarities between successive frames in videos at various levels. We identify *online-pruning* opportunities for inferences, which can also be exploited at *frame-level*, *region-level*, and *pixel-level.*
- We then propose a frame-level motion vector based scheme to leverage the frame-level similarity to opportunistically skip the inference by reusing the compute results memoized by the previous frame. In order to maintain high accuracy, we also propose an adaptive technique to dynamically adjust the reuse window size based on the runtime statistics by comparing the MVs in consequent frames.
- Next, we propose a tile-level inference scheme to enable the region/tile reuse by identifying the critical regions. Since processing these critical regions is in general much lighter-weight than processing a whole frame due to smaller size and less computation, the proposed partial inference is able to minimize the unnecessary computation (on the background/unimportant regions), thereby further speeding up the inference and reducing the energy consumption.
- Then, we implement and experimentally evaluate our proposal on a representative mobile device – the Pixel 3 Phone [20] – and collect detailed experimental results, using 6 different video streams. Our proposed frame-level reuse shows $\approx 53\%$ of the frames to be redundant and hence skips the inference, leading to only less than 1% accuracy loss. Combining the frame-level reuse with the partial inference gives $2.2\times$ performance improvement on average, and up to 80% energy savings, while losing less than 2% accuracy.
- Finally, We compare our approach against four state-of-the-art works (DeepCache [8], Euphrates [9], Potluck [12], and MCDNN [7]). We outperform the MCDNN [7] and Euphrates [9] in terms of accuracy, and Potluck [12] and DeepCache [8] in terms of performance improvement.

II. Background and Related Work

In this section, we start our discussion by explaining a typical DNN execution on mobile devices for video analytics. We then go over potential optimization opportunities that prior works have explored to make DNN inference mobile-friendly.

A. Video Inference on Mobile Platforms

The key difference between a video-based DNN application and other popular DNN inferencing applications like natural language processing (NLP) or speech-to-text is that, the former interacts with video frames which are either captured from the camera or downloaded/streamed from internet and hence, has a strict latency requirement for performing inferencing within the frame deadline. As shown in the "HW" (Hardware) layer in Fig. 1, a typical mobile neural network video inference system has two major hardware components: (i) an SoC with a CPU/GPU/NPU for processing the intensive computations, and an intermediate buffer in DRAM for storing the video frames as well as the intermediate data between layers of DNNs, and (ii) a video decoder communicating with the SoC ,typically via the memory bus. Running on top of the hardware, as shown in the "Application" layer in Fig. 1, the neural network video inference software pipeline can be summarized as follows:

Input: The raw video data is first stored in memory (usually in the H.264/MPEG format). A hardware-based H.264/MPEG decoder decodes the compressed video bitstreams to obtain the original video frames. These frames are then buffered in a memory buffer, waiting for the next stage – NN Inference.

NN Inference: In this stage, the neural network (NN) pipeline takes the decoded video data to perform the inference tasks with the available compute engines (e.g., CPU, GPU, NPU, or any dedicated ASICs). Numerous prior studies (e.g., see [21]–[23] and the references therein) have clearly shown that, regardless of the type of the compute hardware employed, the NN inferences are both compute and memory intensive. As a result, this stage is the main bottleneck in the NN applications.

Output: Following the inference stage, the resulting Feature-Maps (FMs) are used to generate the final tags/bounding-boxes and finally report to the application (e.g., a cow has been identified in the image with 95% confidence).

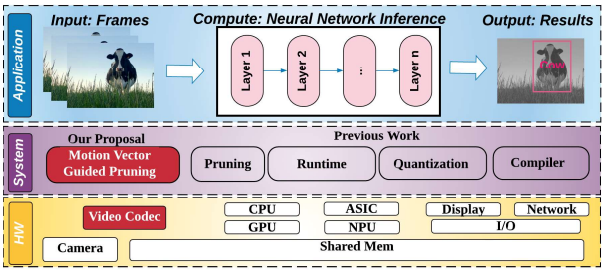


Fig. 1: A DNN inference pipeline on an edge device with optimizations in the application, system, and hardware levels.

B. Prior Work

Prior work for enabling inferences on edge devices have focused on hardware as well as software optimizations, which can be further classified into model compression and pruning, and compiler and runtime support.

1) Hardware Optimizations: Traditionally, CPUs and GPUs have been recruited for DNN inference on mobile phones. Although DNN inference is highly structured and embarrassingly parallel, the limited resources on the mobile devices, alongside the required off-chip data movements, poses a significant challenge leading to higher latency. Several hardware-based approaches such as NPUs [24] or ASICs [25], [26] have been proposed to speedup the inference; however, they only work on a narrow set of specific applications and/or platforms.

2) Model Compression and Pruning: To make DNNs mobile friendly, there have been several works in compressing and pruning large models. While model compression [27]–[29] tries to compress the entire weight matrix while preserving accuracy, pruning goes over individual weights/kernels and drops the unimportant ones. Model compression is typically achieved by tensor decomposition or low rank compression, i.e., by representing the higher dimensional parameter matrix in a compressed low rank form. In contrast, pruning looks at the contribution of each individual weight. While some of the weights contribute more towards accuracy, some contribute less; and the ones contributing less are dropped to reduce the parameter size as well as compute and memory footprints. On the other hand, quantization, which has recently gained a lot of traction, trying to quantize the parameters to lower precision [25]. Combined with specialized hardware support, quantization can reduce the amount of computation, memory footprint, and energy consumption. Note that these compression and pruning approaches can be performed at compile time, and they typically need fine-tuning to retain accuracy.

3) System Support for Exploiting Pixel and Computation Similarities: State-of-the-art proposals such as DeepCache [8], and Euphrates [9] have explored the temporal similarity at runtime for DNN inference on video streams. Therefore, in Sec. V, we perform a detailed comparison of our work against these prior works. As mentioned in Sec. I, DeepCache does not take advantage of frame-wise data reuse opportunities (e.g., reuse the inference result for similar frames). Thus, the latency improvement and energy saving potential of DeepCache can be limited. Additionally, as the step-size of full-inference is fixed, it cannot adaptively update its cache based on the video content. On the other hand, Euphrates makes use of the motion information collected from the Image Signal Processor (ISP), and search the RoIs by combining the MVs of the current frame with the inference result of reference frame, and consequently, decrease the number of inference. Different from these two prior works where approximation decisions are dictated by reuse distance (e.g., the distance between two fully-inferenced frames), Potluck [12] utilized the feature vector extracted from input frames to adaptively trade off computation with reuse of the cached results.

In addition to those described above, several offloading-based techniques have been proposed to speed up the inference on edge. For example, a dynamic RoI encoding is proposed to compress the data volume to be transferred through network [17]. An anchor-point selection algorithm (running in cloud) is also proposed to decide whether to reuse the previous results or not in [30]. Both require the assistance of the cloud, and hence, introduce additional costs and privacy issues.

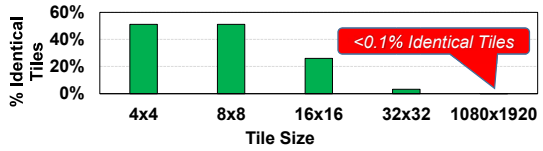
To summarize, even though the prior works discussed above have tried the optimized DNN models, the static compiler/run-

time support and hardware-based enhancements, performance- and energy-efficient execution of DNN pipelines for videos on mobile devices are still open problems. Our main goal in this work is to look deeper into the existing pipeline to dynamically identify better opportunities for optimizations, with minimum changes to the existing software-hardware stack.

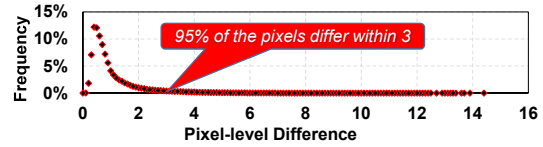
III. Similarity in Video Applications

Due to the limited computing resources and the strict power/energy budget constraints [2], enabling high-quality fast inference on mobile devices is very challenging for DNN applications (e.g., object detection for videos). In fact, the inference of VGG-16 [31], which is a popular DNN model, takes 240 ms to execute on an embedded Adreno 640 [32] GPU, which is far from *real-time*. Moreover, most of the existing solutions targeting such applications demand specialized hardware and/or compiler support, and it is not straightforward for a developer to deploy them without multi-domain expertise across the optimization, compilers and hardware spectrum.

However, in the specific context of video applications, the "temporal continuity nature" of the video data presents itself as an opportunity that is yet to be fully exploited. As a sample work, Euphrates [9] employs MVs to detect small changes between frames and skips unnecessary inferences. However, the number of skipped inferences there is *static*. We believe that, by looking deeper into the frame similarities and identifying reuse opportunities at a "finer level", we can significantly reduce the number of inferences, thereby reduce the burden on the hardware. Further, if we can *dynamically* exploit this opportunistic similarity (i.e., the inference is invoked based on runtime contents), the solution can encompass most vision applications without affecting the current hardware stack.



(a) Similarity study at different tile size across adjacent frames for a video in VIRAT dataset [33].



(b) Distribution of pixel-level difference across two adjacent frames.

Fig. 2: Similarity study.

As discussed in Sec. II, most existing optimizations focus on accelerating the inference processing/computation only and not fully understand the underlying characteristics of the input data, and thus some input-specific optimization opportunities could be easily missed. For example, for the mobile video inference application considered in this paper, the input data are continuous video frames, and can potentially contain rich "similarity" across different frames.

To explore these opportunities, Fig. 2a plots the "pixel-level similarity" between two successive frames in a video. Here, we vary the tile size (an N×N tile is a N×N pixel block) from 4×4 to the entire frame (1080×1920) on the x-axis, and the y-axis gives the fraction of "identical tiles" (when compared pixel-by-pixel) in *two successive frames*. From this figure, we can find that: **1).** Small-size tiles share significant similarities across consecutive frames. E.g., nearly 50% of the tiles are identical in successive frames with a tile size of 8×8. **2).** However, as the tile size increases, tiles in successive frames become more dissimilar. E.g., with 32×32 tiles, only 3% of the tiles are identical. In the extreme case where the tile size equals frame size, only less than 0.1% of the successive frames are identical.

As a result, any approach trying to exploit frame reuse (e.g., skip inferences for similar frames) based on this specific similarity metric (exact pixel match) will not have much scope for optimization. Instead, one may want to consider alternate similarity metrics that can lead to richer reuse opportunities.

The distribution of the absolute differences (deltas) between the pixel values in successive frames plotted in Fig. 2b provide one such opportunity. It can be observed that, while the "exact pixel match based similarity" is scarce at frame level, an alternate similarity based on the "magnitude of pixel differences" is abundant. Specifically, about 95% of the deltas (pixel differences in successive frames) are less than 3. The rest of this paper presents and experimentally evaluates two novel optimization strategies that exploit this similarity.

IV. Proposed Strategies

As discussed in Sec. III, there exist significant similarities between two successive frames (e.g., for more than 95% of the pixels between two adjacent frames) when considering "pixel difference" as the similarity metric. In the following two subsections, targeting inference-based video applications, we present two novel schemes that take advantage of this similarity: *frame-level pruning* and *region-level pruning*.

A. Frame-Level Pruning

1) Shortcomings of Pixel-by-Pixel Comparison: We believe a solution based on exact pixel-by-pixel matching would be too strict and would not be suitable for DNN-based applications. This can be illustrated with a simple use-case scenario. Consider an object detection scenario where a multi-color LED bulb on a Christmas tree identified as an object of interest, glowing red in Frame$_i$, changes to blue in Frame$_{i+1}$. In such a case, as the pixel values of the identified object have changed, Frame$_{i+1}$ cannot reuse the result (LED bulb) from Frame$_i$ even though it should ideally be able to do so in an object detection application. Hence, rather than relying on low-level raw pixel values, most DNN applications leverage high-level features, where "new events" or "motions" make more sense to employ in determining whether we can reuse the results from previous

frame (or skip the current frame). We next discuss the motion vector concept, which is used in our approach.

2) Motion Vectors: The motion vector (MV), which is built upon pixel differences, can be a good candidate to capture the reusability in video analytics. A MV is a 2-D vector that provides an offset from the coordinates in the decoded frame to the coordinates in a reference frame [34], which can be directly obtained from the codec [35] without any post-processing. As opposed to the software-based "optical flow" solution widely used in the computer vision domain [36], collecting the MV from the *codec hardware* is quite light-weight. In fact, our profiling indicates that only tens of μs are needed to generate the MVs for one frame (negligible compared to milliseconds or even seconds that DNN inference takes). We next illustrate 3 common scenarios of leveraging the MVs to capture the reuse opportunities in a DNN-based object detection application.

3) Three Scenarios of Frame-Level Reuse: ⓘ **Moving Object(s)**: As shown in Fig. 3a, one of the most common cases in videos is that the object(s) (which have been identified in previous frames, i.e., Frame-1) move around in the current frame (i.e., Frame-2 , Frame-3). In such scenarios, to explore the reusability exposed by motion vectors, ❶ we first process the full inference in CPU[1] for Frame-1, and ❷ identify the objects as well as their positions (i.e., bounding boxes). ❸ Then for Frame-2, we obtain the its motion vectors from the codec (with a minimal overhead, as discussed in Sec. IV-A2). With such knowledge, for Frame-2, we can observe that the bounding box and the MVs mostly overlap (with an overlap ratio of 0.71 in the left case), which indicates that the objects have barely moved. Thus,we can safely *skip* the inference for Frame-2, and simply reuse the result from Frame-1, as shown in ❺. However, for Frame-3, the motion vectors generated by codec (❻) drift away from the bounding box in Frame-1 (with an overlap ratio of 0.6 in the right case), indicating that the object has moved/shifted a significant distance. Thus, we need to perform *full inference* for Frame-3 to maintain high accuracy for detection, as shown in ❼. Putting all these together, in this scenario, both Frame-1 and Frame-3 employ full inference, whereas the inference for Frame-2 can be skipped, with very little overhead (only 0.5% of the time that full inference takes, refer to Sec. V for more details).

ⓘⓘ **Missing Object(s)**: Another scenario is one in which the object(s), which have not been identified in the previous frames, are detected in the current frame, as shown in Fig. 3b. In this scenario, the area of the MV S_{mv} is similar to or even larger than the area of the smallest bounding box S_{sbb} from the previous frame (in this case, $S_{mv} = 1.3 \times S_{sbb}$). Such large MV indicates that an object (of similar size to that of the other objects which have been already identified), which was supposed to be captured, but failed due to the inaccuracy of DNN models (such as YOLOv4-tiny [37] used by the detection application as discussed in Sec. V). Therefore, in this case, Frame-2 needs to be carefully processed and full inference needs to be employed, as indicated by ❹ in Fig. 3b.

ⓘⓘⓘ **Entering/Exiting Object(s)**: Another scenario is where one or more objects are moving into or out from a frame (refer Frame-2 in Fig. 3c). In this case, although the MV is smaller than the bounding boxes, its position is on the edge, indicating that a new object is entering the frame. As a result, Frame-2 is critical as it reveals new information that has not been exposed before, and thus requires performing full inference for it.

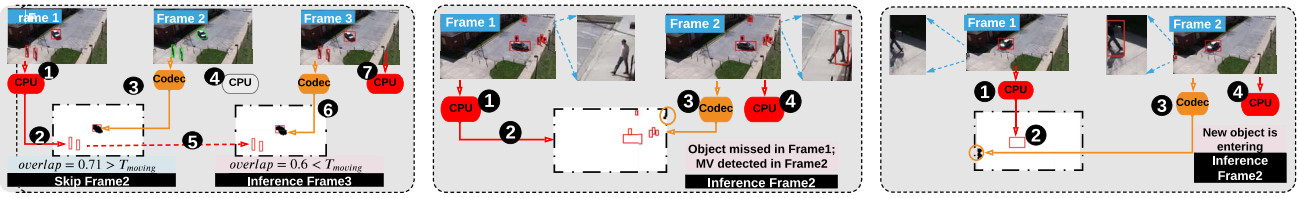
Algorithm 1: Adaptive Frame Level Reuse Algo.

Input : RD: Reuse Distance
Input : BBs: Bounding Boxes in Previous Frames
Input : MVs: Motion Vectors for Current Frame
Output: $Do\_Full\_Inference$: Decisions

```
1  procedure Moving(BBs, mv)                                  // Scenario1
2      if 0 < max{overlapped.area} ≤ T_moving × mv.area then
3          return True
4      else
5          return False
6  procedure Missed(BBs, mv)                                  // Scenario2
7      if mv.area ≥ T_missed × min{BBs.area} then
8          return True
9      else
10         return False
11 procedure Entering/Exiting(mv)                             // Scenario3
12     if mv is at edge then
13         return True
14     else
15         return False
16 procedure Frame_Decision(RD, BBs, MVs)                     // main
17     if RD ≥ RD_upper_bound then
18         return True
19     if MVs == ∅ then
20         return False
21     Fuse and Filter the MVs
22     if BBs == ∅ then
23         if MVs_is_large then
24             return True
25         else
26             return False
27     for mv in MVs do
28         if mv.area > T_area × minBBs.area then
29             if Moving(BBs,mv) or Missed(BBs,mv) then
30                 return True
31         if Entering/Exiting(mv) then
32             return True
33     return False
```

4) Algorithm: To handle the three common scenarios discussed above, we propose our *adaptive* frame level (FL) reuse algorithm to determine whether to invoke full inference (FI) or skip inference (SI) for the current frame – as shown in Line 16 in Algo. 1. First, to ensure minimal accuracy loss, if we have already skipped inference for frames beyond a certain number ($RD_{\text{upper\_bound}}$), we decide to opt for FI (in Line 17). Note that $RD_{\text{upper\_bound}}$ can vary across different use cases from parking lot to busy roads and hence, can be set accordingly

[1]According to [23], only a small portion of DNN inference run on mobile GPUs, thus, in this work, we focus on optimizing the DNN inference on mobile CPUs.



(a) Scenario1: moving. (b) Scenario2: capturing a previously missed object. (c) Scenario3: entering/existing.

Fig. 3: Three scenarios: Scenario1: An existing object is moving. Scenario2: An object was not captured by the previous frame, but captured by the current frame. Scenario3: A new object is entering the frame (or an existing object is exiting).

for different cases. After filtering out the "no change in MV" cases for SI in Line 19 in Algo.1, we next fuse all of the MVs together and filter the small ones (i.e., noises) out (shown in Line 21). Next, we iterate each MV block for Scenario-1, Scenario-2 and Scenario-3 (shown from Line 22 to Line 31). More specifically, if the MV block is relatively large, then we examine whether the object is moving in the current frame or has been missed by the previous frame (in Line 27–28). If that is the case, then full inference needs to be invoked. Otherwise, we further check whether an object is entering to or exiting from the current frame (in Line 30) to perform full inference for it. Our experiments reveal that the proposed FL scheme can skip up to 53% of the frames, with *very less* accuracy loss (0.075%). Moreover, the total overhead of this algorithm is only 0.5% of the overall inference execution latency for a heavy model, as will be discussed later in detail in Sec. V.

B. Region-Level Pruning

As discussed in Sec. IV-A, while our proposed frame level scheme (coarse granularity) enjoys FI skipping opportunities in some cases, we need to perform FI for the remaining cases, to maintain the accuracy. This leads us to our next question – **Can we do better?** To explore the reuse opportunities at a finer granularity, we now dive into a tile/region-level study.

1) Partial Inference: To further explore the computation reuse opportunities, we revisit the first scenario illustrated in Fig. 3a, where the MV for Frame-3 is not highly overlapped with the bounding boxes from Frame-1 (overlap ratio = 0.60, lower than the predefined threshold). This means that the object in question has been moving towards a direction, which triggers a "position change" event. Consequently, to reflect this event, Frame-3 needs to perform a full inference. However, as we can see from ❻ in Fig. 3a, only the bounding boxes (in red) and the MV (in black) are meaningful, and the remaining regions (in white) are similar with those of previous frames, which, intuitively, do not provide as much contribution to detect the "position change" event denoted by the colored regions. This observation reveals an opportunity to perform *partial inference (PI)* by operating only on the bounding boxes and MV regions. Next, we study the following three questions in detail: (i) for which frames can we opt for the PI?, (ii) how to maintain the accuracy?, and finally, (iii) how to do the PI?

2) High-level Idea: To answer the above questions, we propose region-level partial inference (PI) to first determine

Algorithm 2: Region Level Reuse Algo.

Input : RD: Reuse Distance
Input : BBs: Bounding Boxes in Previous Frames
Input : MVs: Motion Vectors for Current Frame
Output: $Flag$: Decisions (Full, Partial, or Skip)
Output: $RoIs$: Regions of Interest

1 **procedure** $Region\_Decision\,(RD,\ BBs,\ MVs)$ `// main`
2 **if** $Frame\_Decision(RD,\ BBs,\ MVs)$ ***is*** $False$ **then**
3 **return** $\{Skip,$ null$\}$
4 **if** $IsScenario2$ ***or*** $IsScenario3$ **then**
5 **return** $\{Full,$ null$\}$
6 **for** mv ***in*** MVs **do** `// only consider Scenario1`
7 **if** $max\{overlapped.area\} \leq T_{moving2} \times mv.area$ **then**
8 **return** $\{Full,$ null$\}$
9 **return** $\{Partial, \bigcup[BBs, MVs]\}$

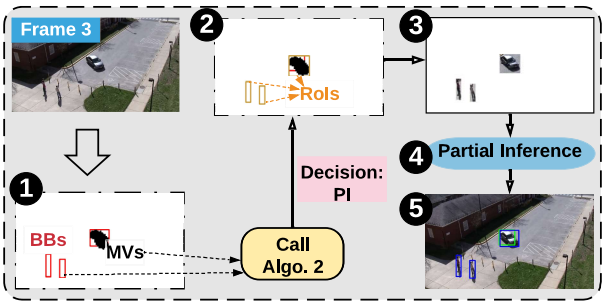


Fig. 4: Main idea in our partial inference scheme.

whether one frame should process the PI, and if so, reuse the cached computation results for "unimportant" regions and only do computation for the "regions of interest" (bounding boxes and MVs), to save computations without much accuracy loss.

To explain the high-level idea behind PI, we reconsider Frame-3 in Scenario-1 discussed in Sec. IV-A, and present our five-step solution in Fig. 4. In Step ❶, same as our FL scheme discussed in Sec. IV-A, the bounding boxes (BBoxes, in red) are extracted by the "full-inferenced" previous frame, and the MVs are obtained from the current frame (Frame-3). Next, based on these inputs including the BBoxes as well as MVs, Algo. 2 is invoked to decide how to process this frame (FI, PI, or SI). As one can see from Line 2 of this algorithm, if the frame has been labeled as "Skip" by Algo. 1, it can be safely bypassed. Additionally, if the frame is in either Scenario-2 or

Scenario-3, it indicates that the object(s) in the current frame are different from the last inference outputs (i.e., "missed" in Scenario-2, or "entering/exiting" in Scenario-3) and hence, requires full inference (refer to Line 5 in Algo. 2). Otherwise, as can be seen from Line 6 to Line 8 in Algo. 2, the overlapped area of each MV and BBox is examined to determine whether the object has moved "too far away" or not; if it has, the "FI" is triggered. Finally, if the frame falls under the category in which objects have not been significantly displaced, then it is labeled as "PI" (as shown in Line 9 in Algo. 2), and the union of the BBoxes and the MVs is reported as the *new* regions of interest (RoIs) (see ❷ in Fig. 4). With these new RoIs, in Step ❸, we now only need to focus on the inputs that map to the RoIs, and omit the other "unimportant" regions. Next, the new inputs are fed into Step ❹ to perform the PI (details are in Sec. IV-B4), and finally, we report the output result to the application, as shown in the blue BBoxes in Step ❺.

Since Algo. 2 has answered the first question (i.e., for a specific frame, what is the inference choice – FI, PI or SI?), we now turn to the second question raised above – how to maintain accuracy? – and study the layers in DNN to explore how to identify which parts are important and which are not.

3) How to Maintain Accuracy?: To preserve accuracy, we back-trace the output feature map (FM) to investigate how the different regions of the input FM affect the the output for convolution layer[2], as shown in Fig. 5 and carefully consider our design decisions. Here, we partition the output FM regions into three categories – ⓐ inner part, ⓑ middle part, and ⓒ outer part. ⓐ is the region where the convolution kernel only multiplies with the pixels in RoIs. In ⓑ, the kernel is multiplied with both the RoI pixels and the non-RoI pixels (background region, i.e., BG), and finally, ⓒ is where the kernel is only multiplied with the pixels inside BG region. Thus, the inner part of the output is only related to the RoIs of input; the middle part is related to both the RoIs and the BG; and the outer part is only related to the BG.

As discussed in Sec. IV-B2, the RoIs are essential to the output results. Thus, the inner part has to be fully inferenced. However, it is *not* desirable to execute the inference on the middle part, since, in order to do so, one needs to prepare an input larger than the inner region (due to the various kernel sizes in the convolution, e.g., 3×3, 5×5 [38]), which eventually turns out to be the FI. Rather than computing, we

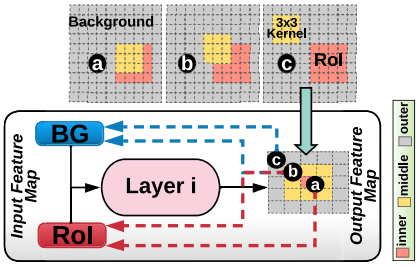


Fig. 5: Back-tracing from *out* to *in* for CONV.

[2]Since CONV layers dominate total computation in DNN inference [8], our partial inference technique is applied only to the CONV layers; the other layers employ full inference.

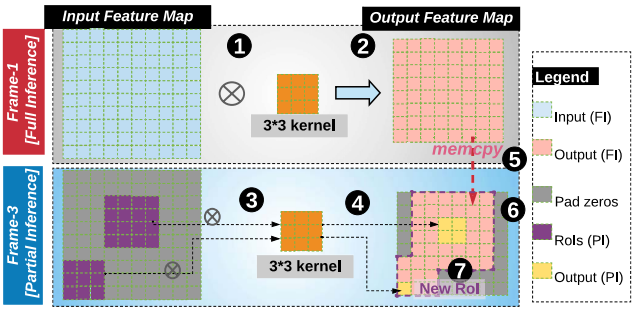


Fig. 6: Partial-inference steps for Frame-3 in Scenario-1.

memoize the feature maps of the middle part in the previous frame, and reuse the data for the current frame. Since the outer part maintains no information for the RoIs, it can be safely discarded.

4) How to Do Partial-Inference?: Next, we revisit the example scenario discussed in Fig. 3a, and give the details of the proposed partial inference (PI) scheme in Fig. 6. Recall that, Frame-1, as the base frame, has to do the full inference (Step ❶), and output the feature maps for each layer (Step ❷). As for Frame-3, only the RoIs (in purple) are passed to the DNN layers (Step ❸), and we generate the corresponding PI-outputs (in yellow) in Step ❹. Next, as discussed in Sec. IV-B3, the FI-output (the neighbors of the partial inference outputs – pink color) from the previous frame is critical to maintain the accuracy of the output for the current frame. Thus, the full inference outputs are accordingly padded around the partial inference outputs via memory copy (Step ❺). For the other regions in the output feature map for this frame, it is safe to simply pad them with zeros (Step ❻). And, finally, we update RoI for the next layer (the region with the purple border in Step ❼). The execution time of the PI for Frame-3 is only 54% of the FI, due to a large amount of computation reduction (achieved by omitting the unimportant regions).

C. Design and Implementation

We designed our both schemes (frame-level reuse in Sec. IV-A and region-level reuse in Sec. IV-B) as plugable modules to the existing compute engines (e.g., CPU, GPU, etc.), as shown in Fig. 7. A *Decision Maker* is placed before the original compute engine (CPU in this example) to dynamically decide how to process the incoming frame, and opportunistically bypass the computation, i.e., either ❶ Full Inference (FI) or ❷ Skip Inference (SI), or reduce the compute by only processing RoIs, when opted for ❸ Partial Inference (PI), with a little overhead (0.5% w.r.t. DNN inference for YOLOv3). To implement these three possible decisions (FI, SI, and PI) that can be made by the *Decision Maker*, the *Inference Engine* in Fig. 7 takes the corresponding actions:

Full Inference: When the incoming frame contains critical information such as new objects entering, a full inference is needed. In this case, the current frame is processed on the CPU to report the final result. Apart from that, the generated

bounding boxes for the object detection task and the feature maps for each layer during the inference are intermediately stored in memory as a "checkpoint". By referring this, it can potentially benefit an inference to be performed later by either completely skipping it (SI) or reducing the computation (PI).

Skip Inference: When the current frame is identified as "clonable" by the previous frame, the CPU is released without any inference execution request. Instead, the previous results (the bounding boxes [including the classes and scores] in the previous frame), which have been memoized before, are directly loaded as the inference result for the current frame.

Partial Inference: In this case, rather than processing the whole frame, only the RoI blocks are fed into the CPU and, with the memoized feature maps from previous frame, the CPU is able to generate the desired result for the current frame as accurate as performing inference on a full frame.

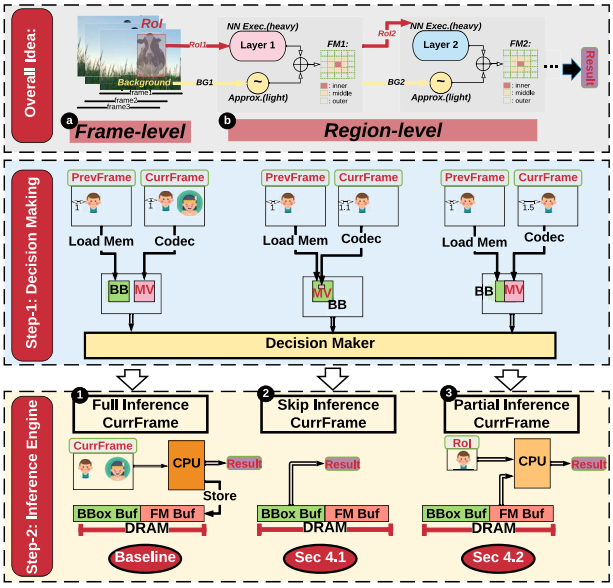


Fig. 7: The proposed frame-level reuse and tile/region-level reuse design blocks implementation; BB/BBox: bounding boxes from the last FI; MV: motion vectors of the current frame; FM: feature maps for each layer from the last FI.

V. Evaluation

Targeting the object detection task on edge devices, we compare our proposed frame-level scheme (*Full-Inference* and *Skip-Inference*, i.e., FI+SI) and region-level scheme (*Full-Inference*, *Skip-Inference* and *Partial-Inference*, i.e., FI+SI+PI) against the *baseline inference*, which performs full inference on every frame, as well as four state-of-the-art runtime techniques (DeepCache [8], Euphrates [9], Potluck [12], and MCDNN [7]), by quantifying the normalized execution time, energy consumption, and accuracy (mean Average Precision, mAP). We first describe the design configurations, experimental platform, datasets, and measurement tools used in this work, and then analyze the collected results.

A. Design configurations

Baseline: We evaluate the baseline video object detection on an edge CPU[1], where every frame is fully inferenced.

FI+SI: We evaluate our FI+SI scheme that skips the inference by utilizing MVs, as discussed in Sec. IV-A. The first frame is considered as the "base", which is always fully inferenced. For the remaining frames however, we invoke Algo. 1 to make the frame-level decision (i.e., either do the full inference or skip)[3]. When skipping, we can simply bypass the inference task by *reusing* the inference result from the last frame; otherwise (i.e., if cannot skip), the full inference has to be performed for the current frame, in the same fashion as in the baseline.

FI+SI+PI: Different from the above FI+SI scheme, in this scheme, the region level decision-making Algo. 2 is invoked to decide among the execution choices for the incoming frames – including Skipping, Full-Inference, and Partial-Inference.

B. Experimental Platform and Datasets

Platforms: We used the Google Pixel 3 Android Phone [20] as our experimental platform. This device consists of a 64-Bit Octa-Core, a Qualcomm Snapdragon 845 SoC, [39], a 4GB LPDDR4X memory, and a 64GB storage. We implemented our FI and PI on top of the ncnn library [5].

mean Average Precision (mAP): The accuracy of the DNN models for object detection can be quantified by mAP [40], to evaluate how well the estimated results match the ground-truth. To get mAP, the precision for each class is first calculated across all of the Intersection over Union (IoU [41]) thresholds, and the final mAP will be the averaged precision of all classes (the higher, the better). The IoU threshold is set as 0.5 (a widely used value in mAP calculation) in our evaluations.

Datasets: We use three published video datasets (VIRAT [33], EPFL [42] and CAMPUS [43]), to study the performance and energy behavior across different videos. The important features of these six videos[4] are summarized in Table II.

Neural Network Models: We examine two DNN models in our experiments: YOLOv3 [44] and YOLOv4-tiny [37]. The input shape for both of them is $1 \times 416 \times 416 \times 3$. The former one is a quite heavy model, with 106 layers and 65.86 Bn FLOPS, whereas the latter one is a lighter model, with 38 layers and 6.94 Bn FLOPS. YOLOv3 achieves 55.3% mAP on average, whereas YOLOv4-tiny achieves 40.2% [45] mAP.

C. Results

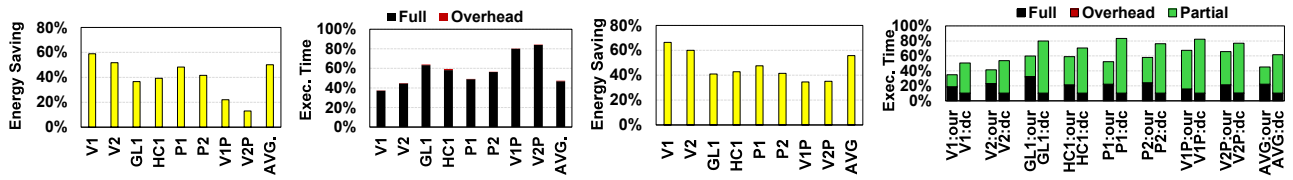
We present and compare the execution latency and energy consumption (via BatteryManager API in Android Studio) when performing inference for each video under the three configurations described in Sec. V-A, as well as the mAP

[3]We experimentally set $T_{moving} = 0.8$, $T_{missed} = 0.8$, $T_{area} = 0.25$ and $RD_{upper\_bound} = 10$ in Algo. 1; $T_{moving2} = 0.3$ in Algo. 2

[4]The ground-truth annotations for the EPFL and CAMPUS datasets were not available to us; so, in this work, we primarily focused on the VIRAT dataset to evaluate the accuracy.

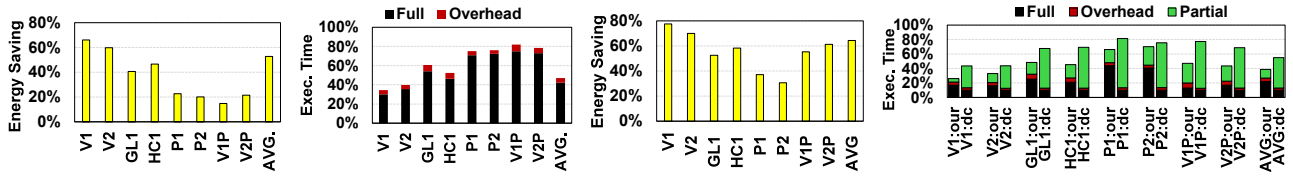
TABLE II: Salient features of the six videos used in this study.

Videos	# Frames	# Avg. Objects	Static/Dynamic	Object/Full Frame	Description
V1 [33]	20655	Medium	Medium	Small	A few people and cars move in the parking lot
V2 [33]	9075	Medium	Medium	Small	More activities of people in the parking lot
GL1 [43]	6477	More	Dynamic	Medium	More activities of people and cars in the parking lot
HC1 [43]	6000	More	Dynamic	Medium	People do exercises in a garden
P1 [42]	3915	Less	Dynamic	Small-Large	Several people walk around in a room
P2 [42]	2955	Medium	Dynamic	Small-Large	More people than P1 in the room



(a) FI+SI energy saving (b) FI+SI exec. time (c) FI+SI+PI energy saving (d) FI+SI+PI exec. time

Fig. 8: Performance and energy improvements for YOLOv3 w.r.t. the baseline; `XX:our` are the results achieved by our proposed FI+PI+SI scheme; `XX:dc` are the results for DeepCache [8]; `V1P` and `V2P` are part of V1 and V2, respectively; and `AVG` is the weighted average (weight is the # of frames in each video).



(a) FI+SI energy saving (b) FI+SI exec. time (c) FI+SI+PI energy saving (d) FI+SI+PI exec. time

Fig. 9: Performance and energy improvements for YOLOv4-tiny w.r.t. the baseline; Region Level Reuse Scheme (e.g., FI+SI+PI) has better performance, because this "shallow" model benefits more from partial inference.

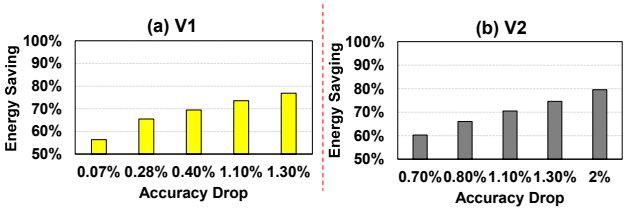


Fig. 10: Tradeoff between accuracy drop and energy saving for (a) V1 and (b) V2 picked from the VIRAT [33] dataset. This shows how the proposed "adaptive" solution can potentially save more energy with different thresholds in Algo. 2.

to evaluate the accuracy, and plot the experimental results in Fig. 8 and Fig. 9. Note that these latency and energy results are *normalized* with respect to the baseline (FI for all frames). From these results, we can make the following observations.

1) Overall Execution Latency: On average, our FI+SI scheme can reduce 52% of the execution time for YOLOv3, and 53% for YOLOv4-tiny, compared to the baseline. On the other hand, our FI+SI+PI scheme can save 55%/61% of the execution time for YOLOv3/YOLOv4-tiny. More specifically, as shown in Fig. 8b and Fig. 8d, the overheads (due to the decision making module in Algo. 1 and Algo. 2) introduced by our proposal amount to only 0.5% of the end-to-end execution latency (the baseline YOLOv3 inference). Also, the PI technique consumes 23% of the execution time in YOLOv3.

Similarly, the overhead introduced by Algo. 1 to YOLOv4-tiny is not too much – 5% of the baseline execution latency, as shown in Fig. 9b. Such small amount of overhead indicates that our decision-making logic is efficient and light-weight.

2) Energy Savings: As shown in Fig. 8a, the YOLOv3 inference with the proposed FI+SI scheme only consumes 50% energy on average, with respect to the baseline, due to the fact that 53% of the inferences can be skipped. The PI technique can further save 6% more energy on average, as shown in Fig. 8c. On the other hand, the YOLOv4-tiny inference with the FI+SI scheme saves 53% energy w.r.t. the baseline, as shown in Fig. 9a. Moreover, when adopting the FI+SI+PI, compared with the FI+SI, around 12% more energy can be saved, as shown in Fig. 9c. These additional energy savings come from the computational reduction by partial inference, rather than computing for the entire frame.

TABLE III: mAP (%) comparison with the baseline

	V1	V2	V3	V4
YOLOv3	51.8	55.4	42.0	59.5
YOLOv3 w/ FI+SI	51.8	55.4	41.8	59.4
YOLOv3 w/ FI+SI+PI	50.3	54.1	40.6	58.0
YOLOv4-tiny	31.8	49.3	31.3	33.5
YOLOv4-tiny w/ FI+SI	31.6	49.2	31.4	33.3
YOLOv4-tiny w/ FI+SI+PI	31.4	49.1	30.8	33.0

3) mAP: To study the accuracy impact of our proposed SI and PI schemes, we summarize the mAP for our two models (YOLOv3 and YOLOv4-tiny) using four videos from VIRAT dataset [33] in Table III. One can observe that the mAP in our FI+SI scheme only drops by 0.075% on average for YOLOv3, and 0.1% for YOLOv4-tiny – both w.r.t. the baseline. With the FI+SI+PI scheme on the other hand, the mAP drops 1.4% in YOLOv3, and 0.4% for YOLOv4-tiny. These results clearly show that our proposal is able to maintain high accuracy.

4) Model-Specific Analysis: To study how the inference behavior changes across different DNN models, we next compare the performance and the energy consumption of two DNN models used in this work (YOLOv3 and YOLOv4-tiny), and plot the results in Fig. 8 and Fig. 9. First, by comparing YOLOv3 (see Fig. 8b) and YOLOv4-tiny (see Fig. 9b), we can observe that YOLOv4-tiny spends less time on inference than YOLOv3 (42% versus 47%). However, the overhead this scheme brings when applied to YOLOv3 (0.5% w.r.t. the baseline) is smaller than that in YOLOv4-tiny (4.8%), resulting in a similar reduction in the overall execution time. Note that, the reason why the overhead in YOLOv4-tiny is higher is because, YOLOv4-tiny is already a very light-weight model, and consequently, the time spent on decision making is not negligible compared to the extremely fast inference it performs.

Additional energy and latency can be saved by the FI+SI+PI scheme, as Fig. 8c-8d and Fig. 9c-9d show, i.e., 55% latency and 56% energy saving in YOLOv3, and 61% latency and 64% energy saving in YOLOv4-tiny. The reason why YOLOv4-tiny saves more is that the PI benefits more in a "shallow" model with a relatively larger room to skip. For example, YOLOv3 takes, on average, $0.75\times$ of the baseline latency to perform the PI on a frame in video HC1 [43], whereas YOLOv4-tiny takes only $\sim 0.48\times$ of the baseline latency.

5) Video-Specific Analysis: To investigate how the performance and energy behavior varies with video salient features such as the object count, objects' motions, the fraction of area occupied by objects in an entire frame, etc., we have studied the six videos summarized in Table II. V1 and V2 are parking lot videos. V1 has fewer objects and less movements, and thus, compared to V2, the savings on execution time and energy consumption for V1 are slightly higher, as shown in Fig. 8b and Fig. 8a. Also, V1 can save 7% more execution time and 6% more energy than V2 when using the FI+SI+PI scheme.

GL1 and HC1 are other two videos from the CAMPUS [43] dataset. GL1 is also a parking lot video, but with many more objects and movements than V1 and V2. HC1 on the other hand is a garden video, with people walking around, riding bike, etc. These two videos are much more active (dynamic) than V1/V2, meaning that they contain less reuse opportunities than V1/V2. Thus, compared to V1 and V2 whose latencies can be decreased by 62% and 55% respectively with FI+SI as shown in Fig. 8b, the latencies for GL1 and HC1 only decrease by around 38%, while the energy saving is about 37%, which is lower than the corresponding savings for V1 (59%) and V2 (53%). Additionally, even with our proposed PI technique, the improvements for GL1 and HC1 with YOLOv3 model are not very significant (e.g., the latency reduction is improved by 4% for GL1 as shown in Fig. 8d). This is because, compared with the whole frame, RoIs in these two videos are large (e.g., ratio of RoI/whole frame is 4x of that ratio in V1), which increases the computation for the PI and thus decreases the its benefits.

Finally, P1 and P2 are two videos with several people walking around a room. As the RoIs size increases along the video (e.g., ratio of RoIs/whole frame increases from 8% in frame20 to 53% in frame2470), the benefits from PI become increasingly lower, resulting in similar patterns for FI+SI and FI+SI+PI. Besides these six videos, we also picked two *slices* from V1 and V2 (i.e., V1P and V2P) to show the effectiveness of our PI technique. In these two slices, the objects keep moving. Thus, most of the frames will perform FI under FI+SI, as shown in Fig. 8b. The latency is only decreased about 18%, while the energy saving is 22% and 13%, as shown in Fig. 8a. With the PI scheme, as shown in Fig. 8d and Fig. 8c, the latency is further decreased by 13% (V1P) and 19% (V2P), whereas the energy saving is increased by 12% and 22%. The improvement brought by PI for YOLOv4-tiny model is more significant as shown in Fig. 9c and Fig.9d, where the latency is further decreased by 36% (V1P) and 35% (V2P), while the energy saving is increased by 41% and 40%, indicating that our PI technique is quite effective.

6) Tradeoffs between Accuracy and Energy Consumption: So far in our evaluation, we wanted to minimize the accuracy impact (see Table III). However, as an alternate design principle, one may want to relax this accuracy constraint and thus save more energy. We used Pytorch [46] to profile the accuracy behavior of two videos picked from VIRAT [33] dataset, and show that our proposal can adaptively support such alternate design choices. More specifically, the <accuracy-loss, energy-savings> pairs are plotted in Fig. 10. From Fig. 10(a), one can observe that, for V1, the energy saving is about 56% with only 0.07% accuracy drop; however, if the application is willing/tolerant to live with 1.3% accuracy drop, then 21% more energy can be saved (amounting to 77% energy reduction compared to the baseline). A similar trend can also be observed in V2, as shown in Fig. 10(b). These results indicate that our proposal is flexible/adaptive with different design preferences.

D. Comparison against Prior Work

As discussed in Sec. II, DeepCache [8] also exploits the similarity between continuous frames at runtime, and decreases the computation via memory copies. However, it misses the opportunities of frame-level data reuse, and hence needs to perform inference for each and every frame. Thus, as shown

TABLE IV: Comparison against Potluck and MCDNN

	mAP	Latency	Energy
MCDNN	36.9%	33%	35%
Potluck	51.6%	63%	61%
This Work	50.3%	35%	34%

in Fig. 8d and Fig. 9d, DeepCache can only save 38% and 45% on execution time for YOLOv3 and YOLOv4-tiny, respectively, which are less than both FI+SI (e.g., 52% for YOLOv3; 53% for YOLOv4-tiny) and FI+SI+PI (e.g., 55% for YOLOv3; 61% for YOLOv4-tiny) schemes.

Another hardware-based optimization has been proposed in Euphrates [9], as discussed in Sec. II-B3. We acknowledge that, compared to our proposal, Euphrates yields around 10% additional energy savings when the window size is set to be as large as 8 (i.e., always skipping 7 frames). We want to emphasize however that, such static (pre-defined) window-size works well only when there are few movements in the videos. On the other hand, when the video has more dynamic behavior, this static skipping jeopardizes the quality of applications which demand high accuracy. To give an example, we selected a segment of frames (i.e., `Frame#6750` to `Frame#6850`) from V1 [33], in which the objects move more aggressively than other segments. In this scenario, our scheme figures out that more movements exist from large MV blocks, and dynamically adjusts the inference decisions. Hence, our scheme does not lose any accuracy, and still saves 43% energy. However, due to the static window size employed, Euphrates [9] leads to as much as 6% accuracy drop in this video segment, which is hardly acceptable by most applications.

Apart from the above mentioned optimizations at the layer-level (DeepCache) and the frame-level (Euphrates), recall that, in Table I, we indicated (in the last column) the Decision Making Logic (DM) the previously proposed optimization strategies employ. Now, in Table IV, we present the mAP, latency and energy efficiency results with those prior schemes. Rather than capturing the reuse opportunities in raw input data, Potluck [12] first extracts the feature vector (i.e., a vector generated from input image, such as SURF [47], HoG [48], Down-sampling [16], etc.) as *key*, and then caches the corresponding inference results (as *value*) for further reuse. We also conducted experiments with a strategy mimicking Potluck [12] on V1 [33] by employing the down-sampled image as the feature vector. As can be observed from Table IV, Potluck [12] provides very good accuracy – 51.6%, which is slightly better than 50.3% provided by our approach. However, it results in almost twice the latency and energy consumption with respect to our approach. This high latency/energy consumption is due to the imprecise down-sampled feature vector, and this further indicates that, a strategy that is based on feature vector memoization can still miss a lot of optimization opportunities.

Different from the three prior works discussed above, which mainly focus on one model, MCDNN targets a multi-model system and proposes a runtime scheduler to improve the accuracy as much as possible within a limited energy budget. We also tested this idea in our framework (with YOLOv3 [44] and YOLOv4-tiny [37] as the available models) and set the energy budget for the scheduler to be the total energy consumption resulting from our approach. As can be seen from Table IV, MCDNN yields similar latency/energy savings with our approach, but with significantly lower accuracy (36.9%). This is mainly because, in MCDNN, the scheduler tends to choose YOLOv4-tiny due to the low energy budget.

E. Future Work

As shown in Sections IV-A and IV-B, our inference decision (FI, SI, or PI) is made by comparing the overlap between the MVs and the previous frame's BBoxes, or the ratio of the MVs size to the previous BBoxes, with a preset thresholds. Although we have an intuitive feeling on the how the thresholds in Algo. 1 and Algo. 2 affect the accuracy and performance, e.g., a larger T_{moving} in Algo. 1 has a preference on FI and is accuracy-friendly, while decreasing T_{moving} will skip more frames and improve the performance, the specific impact of the selected thresholds on accuracy drop, performance improvement, and energy reduction deserves further study.

Also, as discussed in Sec. V-C, the overheads brought by our decision making amount to 5% of the DNN inference time for YOLOv4-tiny when running on CPU, which reduces the performance and energy savings for our proposed schemes. If we can deploy the decision making logic on a *custom hardware* with negligible overhead, our proposed techniques would be more effective when targeting light DNN models.

VI. Concluding Remarks

Pushing DNN-assisted video analysis into edge is the current trend in applications like surveillance, assisted surgery, and VR/AR [23]. Along this trend, prior approaches have targeted improving either accuracy or performance. In contrast, this paper revisits the <accuracy, energy, performance> design space, and tunes the design knobs adaptively with the changing constraints of applications over time. Specifically, we propose and evaluate two schemes (for skipping the inference, and bypassing the compute for unimportant regions), to improve performance and save energy. Our evaluations indicate $2.2\times$ speedup and 56% energy saving over the baseline setting. Further, relaxing the accuracy loss tolerance to 2%, we can save up to 80% energy. Additionally, the experimental analysis indicates that our approach outperforms the state-of-the-art work with respect to accuracy and/or performance/energy savings.

Acknowledgment

We thank the anonymous reviewers for their helpful feedback and suggestions towards improving the paper content. This research is supported in part by NSF grants #1931531, #1955815, #2116962, #2122155 and #2028929.

References

[1] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 289–304.

[2] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-Based Weight Pruning," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, p. 907–922.

[3] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices," *arXiv preprint arXiv:1909.05073*, pp. 5117–5124, 2019.

[4] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[5] Tencent, "NCNN," "https://github.com/Tencent/ncnn", 2017.

[6] Alibaba, "MNN: Mobile Neural Network," "https://github.com/alibaba/MNN", 2019.

[7] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016, p. 123–136.

[8] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "DeepCache: Principled Cache for Mobile Deep Vision," in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018, p. 129–144.

[9] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision," in *Proceedings of the International Symposium on Computer Architecture*, 2018, p. 547–560.

[10] M. Riera, J.-M. Arnau, and A. González, "Computation Reuse in DNNs by Exploiting Input Similarity," in *Proceedings of the International Symposium on Computer Architecture*, 2018, p. 57–68.

[11] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-Based Deep Learning Framework for Continuous Vision Applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, p. 82–95.

[12] P. Guo and W. Hu, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," p. 271–284, 2018.

[13] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices," in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2017, p. 409–421.

[14] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "A Power-Efficient and High Performance FPGA Accelerator for Convolutional Neural Networks: Work-in-Progress," in *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*, 2017.

[15] J. Wang, J. Lin, and Z. Wang, "Efficient Hardware Architectures for Deep Convolutional Neural Network," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1941–1953, 2017.

[16] Aashish Chaubey, "Downsampling and Upsampling of Images - Demystifying the Theory," "shorturl.at/rCMPU", 2020.

[17] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-Time Object Detection for Mobile Augmented Reality," in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[18] H. Jiang, A. Sarma, J. Ryoo, J. B. Kotra, M. Arunachalam, C. R. Das, and M. T. Kandemir, "A learning-guided hierarchical approach for biomedical image segmentation," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*, 2018, pp. 227–232.

[19] H. Jiang, A. Sarma, M. Fan, J. Ryoo, M. Arunachalam, S. Naveen, and M. T. Kandemir, "Morphable convolutional neural network for biomedical image segmentation," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1522–1525.

[20] Google, "Pixel Phone Hardware Tech Specs," "https://bit.ly/397dCUB".

[21] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing CNN Model Inference on CPUs," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, 2019, p. 1025–1040.

[22] M. Motamedi, D. D. Fong, and S. Ghiasi, "Fast and Energy-Efficient CNN Inference on IoT Devices," *CoRR*, 2016.

[23] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine Learning at Facebook: Understanding Inference at the Edge," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.

[24] D. A. Palmer and M. Florea, "Neural Processing Unit," 2014, uS Patent 8,655,815.

[25] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "DRQ: Dynamic Region-Based Quantization for Deep Neural Network Acceleration," in *Proceedings of the International Symposium on Computer Architecture*, 2020, p. 1010–1021.

[26] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," in *Proceedings of the International Symposium on Computer Architecture*, 2018, p. 764–775.

[27] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2015.

[28] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," *arXiv preprint arXiv:1511.06530*, 2015.

[29] C. Louizos, K. Ullrich, and M. Welling, "Bayesian Compression for Deep Learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 3290–3300.

[30] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han, "NEMO: Enabling Neural-Enhanced Video Streaming on Commodity Mobile Devices," in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2020.

[31] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] Klaus Hinum, "Qualcomm Adreno 640," "shorturl.at/btCH3", 2018.

[33] Kitware, Inc., "The VIRAT Video Dataset," "https://viratdata.org", 2011.

[34] A. Tamhankar and K. R. Rao, "An overview of H.264/MPEG-4 Part 10," in *Proceedings EC-VIP-MC 2003. 4th EURASIP Conference focused on Video/Image Processing and Multimedia Communications (IEEE Cat. No.03EX667)*, 2003, pp. 1–51 vol.1.

[35] Google Developers, "MediaCodec," "shorturl.at/mCHV4", 2021.

[36] S. S. Beauchemin and J. L. Barron, "The Computation of Optical Flow," *ACM Comput. Surv.*, p. 433–466, 1995.

[37] Jacob Solawetz, Samrat Sahoo, "Train YOLOv4-tiny on Custom Data - Lightning Fast Object Detection ," "shorturl.at/vCSVW", 2020.

[38] MYO NeuralNet , "Calculating the Output Size of Convolutions and Transpose Convolutions," "shorturl.at/ioLRV", 2020.

[39] Qualcomm Technologies Inc., "Snapdragon 845 Mobile Platform," "shorturl.at/fouyP", 2018.

[40] L. Liu and M. T. Zsu, *Encyclopedia of Database Systems*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[41] A. Rosebrock, "Intersection over Union (IoU) for Object Detection," "shorturl.at/gszOR", 2016.

[42] CVLAB in EPFL, "Multi-camera Pedestrians Video," "shorturl.at/zDY25".

[43] Y. Xu, X. Liu, L. Qin, and S.-C. Zhu, "Cross-View People Tracking by Scene-Centered Spatio-Temporal Parsing," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, p. 4299–4305.

[44] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, 2018.

[45] B.-G. Han, J.-G. Lee, K.-T. Lim, and D.-H. Choi, "Design of a Scalable and Fast YOLO for Edge-Computing Devices," *Sensors*, 2020.

[46] PyTorch Development Team, "PyTorch," "https://github.com/pytorch/pytorch", 2016.

[47] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, 2006, pp. 404–417.

[48] F. Suard, A. Rakotomamonjy, A. Bensrhair, and A. Broggi, "Pedestrian detection using infrared images and histograms of oriented gradients," in *2006 IEEE Intelligent Vehicles Symposium*, 2006, pp. 206–212.