Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision

Yuhao Zhu[1] Anand Samajdar[2] Matthew Mattina[3] Paul Whatmough[3]

[1]University of Rochester
[2]Georgia Institute of Technology
[3]ARM Research

Abstract

Continuous computer vision (CV) tasks increasingly rely on convolutional neural networks (CNN). However, CNNs have massive compute demands that far exceed the performance and energy constraints of mobile devices. In this paper, we propose and develop an algorithm-architecture co-designed system, Euphrates, that simultaneously improves the energy-efficiency and performance of continuous vision tasks.

Our key observation is that changes in pixel data between consecutive frames represents visual motion. We first propose an algorithm that leverages this motion information to relax the number of expensive CNN inferences required by continuous vision applications. We co-design a mobile System-on-a-Chip (SoC) architecture to maximize the efficiency of the new algorithm. The key to our architectural augmentation is to *co-optimize different SoC IP blocks in the vision pipeline collectively*. Specifically, we propose to expose the motion data that is naturally generated by the Image Signal Processor (ISP) early in the vision pipeline to the CNN engine. Measurement and synthesis results show that Euphrates achieves up to 66% SoC-level energy savings (4× for the vision computations), with only 1% accuracy loss.

1. Introduction

Computer vision (CV) is the cornerstone of many emerging application domains, such as advanced driver-assistance systems (ADAS) and augmented reality (AR). Traditionally, CV algorithms were dominated by hand-crafted features (e.g., Haar [109] and HOG [55]), coupled with a classifier such as a support vector machine (SVM) [54]. These algorithms have low complexity and are practical in constrained environments, but only achieve moderate accuracy. Recently, convolutional neural networks (CNNs) have rapidly displaced hand-crafted feature extraction, demonstrating significantly higher accuracy on a range of CV tasks including image classification [104], object detection [85, 97, 99], and visual tracking [56, 91].

This paper focuses on *continuous vision* applications that extract high-level semantic information from *real-time* video streams. Continuous vision is challenging for mobile architects due to its enormous compute requirement [117]. Using object detection as an example, Fig. 1 shows the compute requirements measured in Tera Operations Per Second (TOPS) as well as accuracies between different detectors under 60 frames per second (FPS). As a reference, we also overlay the 1 TOPS line, which represents the peak compute capability that today's CNN accelerators offer under a typical 1 W mobile power budget [21, 41]. We find that today's CNN-based approaches such as YOLOv2 [98], SSD [85], and Faster R-CNN [99] all have at least one order of magnitude higher compute requirements than accommodated in a mobile device. Reducing the CNN complexity (e.g., Tiny YOLO [97], which is a heavily truncated version of YOLO with 9/22 of its layers) or falling back to traditional hand-crafted features such as Haar [61] and HOG [113] lowers the compute demand, which, however, comes at a significant accuracy penalty.

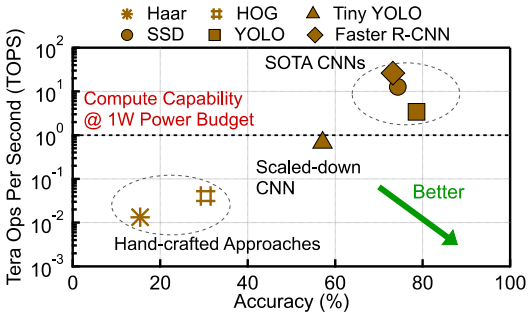


Fig. 1: Accuracy and compute requirement (TOPS) comparison between object detection techniques. Accuracies are measured against the widely-used PASCAL VOC 2007 dataset [32], and TOPS is based on the 480p (640×480) resolution common in smartphone cameras.

The goal of our work is to improve the compute efficiency of continuous vision with small accuracy loss, thereby enabling new mobile use cases. The key idea is to exploit the motion information inherent in real-time videos. Specifically, today's continuous vision algorithms treat each frame as a standalone entity and thus execute an entire CNN inference on every frame. However, pixel changes across consecutive frames are not arbitrary; instead, they represent visual object motion. We propose a new algorithm that leverages the temporal pixel motion to synthesize vision results on many frames with little computation while avoiding expensive CNN inferences.

Our main architectural contribution in this paper is to co-

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00052

design the mobile SoC architecture to support the new algorithm. Our SoC augmentations harness two insights. First, we can greatly improve the compute efficiency while simplifying the architecture design by *exploiting the synergy between different SoC IP blocks*. Specifically, we observe that the pixel motion information is naturally generated by the ISP early in the vision pipeline owing to ISP's inherent algorithms, and thus can be obtained with little compute overhead. We augment the SoC with a lightweight hardware extension that exposes the motion information to the vision engine. In contrast, prior work extracts motion information manually, either offline from an already compressed video [45, 116], which does not apply to real-time video streams, or by calculating the motion information at runtime – at a performance cost [73, 101].

Second, although the new algorithm is light in compute, implementing it in software is energy-inefficient from a system perspective because it would frequently wake up the CPU. We believe that always-on continuous computer vision should be task-autonomous, i.e., free from interrupting the CPU. Hence, we introduce the concept of a *motion controller*, which is a new hardware IP that autonomously sequences the vision pipeline and performs motion extrapolation—all without interrupting the CPU. The motion controller's microarchitecture resembles a micro-controller, and thus incurs very low design cost.

We develop Euphrates, a proof-of-concept system of our algorithm-SoC co-designed approach. We evaluate Euphrates on two tasks, object tracking and object detection, that are critical to many continuous vision scenarios such as ADAS and AR. Based on real hardware measurements and RTL implementations, we show that Euphrates doubles the object detection rate while reducing the SoC energy by 66% at the cost of less than 1% accuracy loss; it also achieves 21% SoC energy saving at about 1% accuracy loss for object tracking.

In summary, we make the following contributions:

- To our knowledge, we are the first to exploit sharing motion data across the ISP and other IPs in a mobile SoC.
- We propose the Motion Controller, a new IP that autonomously coordinates the vision pipeline during CV tasks, enabling "always-on" vision with very low CPU load.
- We model a commercial mobile SoC, validated with hardware measurements and RTL implementations, and demonstrate significant energy savings with little accuracy loss.

The remainder of the paper is organized as follows. Sec. 2 introduces the background. Sec. 3 and Sec. 4 describe the motion-based algorithm and the co-designed architecture, respectively. Sec. 5 describes the evaluation methodology, and Sec. 6 quantifies the benefits of Euphrates. Sec. 7 discusses limitations and future developments. Sec. 8 puts Euphrates in the context of related work, and Sec. 9 concludes the paper.

2. Background and Motivation

We first give an overview of the continuous vision pipeline from the software and hardware perspectives (Sec. 2.1). In particular, we highlight an important design trend in the vision frontend where ISPs are increasingly incorporating motion estimation, which we exploit in this paper (Sec. 2.2). Finally, we briefly describe the block-based motion estimation algorithm and its data structures that are used in this paper (Sec. 2.3).

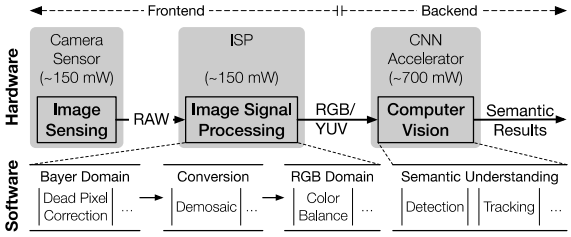


Fig. 2: A typical continuous computer vision pipeline.

2.1. The Mobile Continuous Vision Pipeline

The continuous vision pipeline consists of two parts: a frontend and a backend, as shown in Fig. 2. The frontend prepares pixel data for the backend, which in turn extracts semantic information for high-level decision making.

The frontend uses (off-chip) camera sensors to capture light and produce RAW pixels that are transmitted to the mobile SoC, typically over the MIPI camera serial interface (CSI) [20]. Once on-chip, the Image Signal Processor (ISP) transforms the RAW data in the Bayer domain to pixels in the RGB/YUV domain through a series of algorithms such as dead pixel correction, demosacing, and white-balancing. In architecture terms, the ISP is a specialized IP block in a mobile SoC, organized as a pipeline of mostly stencil operations on a set of local SRAMs ("line-buffers"). The vision frontend typically stores frames in the main memory for communicating with the vision backend due to the large size of the image data.

The continuous vision backend extracts useful information from images through semantic-level tasks such as object detection. Traditionally, these algorithms are spread across DSP, GPU, and CPU. Recently, the rising compute intensity of CNN-based algorithms and the pressing need for energy-efficiency have urged mobile SoC vendors to deploy dedicated CNN accelerators. Examples include the Neural Engine in the iPhoneX [4] and the CNN co-processor in the HPU [29].

Task Autonomy During continuous vision tasks, different SoC components autonomously coordinate among each other with minimal CPU intervention, similar to during phone calls or music playback [90]. Such a task autonomy frees the CPU to either run other OS tasks to maintain system responsiveness, or stay in the stand-by mode to save power. As a comparison, the typical power consumption of an image sensor, an ISP, and a CNN accelerator combined is about 1 W (refer to Sec. 5.1 for more details), whereas the CPU cluster alone can easily consume over 3 W [68, 82]. Thus, we must maintain task autonomy by minimizing CPU interactions when optimizing energy-efficiency for continuous vision applications.

Object Tracking and Detection This paper focuses on two continuous vision tasks, object tracking and detection, as

they are key enablers for emerging mobile application domains such as AR [16] and ADAS [24]. Such CV tasks are also prioritized by commercial IP vendors, such as ARM, who recently announced standalone Object Detection/Tracking IP [9].

Object tracking involves localizing a moving object across frames by predicting the coordinates of its bounding box, also known as a region of interest (ROI). Object detection refers to simultaneous object classification and localization, usually for multiple objects. Both detection and tracking are dominated by CNN-based techniques. For instance, the state-of-the-art object detection network YOLO [97] achieves 37% higher accuracy than the best non-CNN based algorithm DPM [65]. Similarly, CNN-based tracking algorithms such as MDNet [91] have shown over 20% higher accuracy compared to classic hand-crafted approaches such as KCF [72].

2.2. Motion Estimation in ISPs

Perceived imaging quality has become a strong product differentiator for mobile devices. As such, ISPs are starting to integrate sophisticated computational photography algorithms that are traditionally performed as separate image enhancement tasks, possibly off-line, using CPUs or GPUs. A classic example is recovering high-dynamic range (HDR) [57], which used to be implemented in software (e.g., in Adobe Photoshop [26]) but is now directly built into many consumer camera ISPs [8, 10, 13, 30].

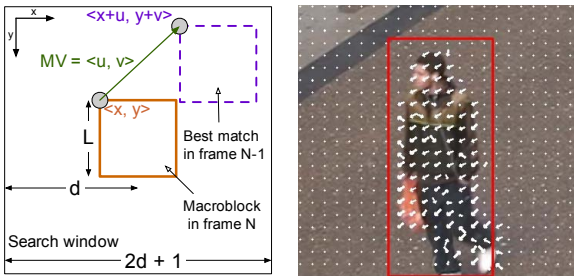
Among new algorithms that ISPs are integrating is *motion estimation*, which estimates how pixels move between consecutive frames. Motion estimation is at the center of many imaging algorithms such as temporal denoising, video stabilization (i.e., anti-shake), and frame upsampling. For instance, a temporal denoising algorithm [75, 84] uses pixel motion information to replace noisy pixels with their noise-free counterparts in the previous frame. Similarly, frame upsampling [52] can artificially increase the frame rate by interpolating new frames between successive real frames based on object motion.

Motion-based imaging algorithms are traditionally performed in GPUs or CPUs later in the vision pipeline, but they are increasingly subsumed into ISPs to improve compute efficiency. Commercial examples of motion-enabled camera ISPs include ARM Mali C-71 [8], Qualcomm Spectra ISP [28], and products from Hikvision [18], ASICFPGA [10], PX4FLOW [27], Pinnacle Imaging Systems [13], and Centeye [34], just to name a few based on public information.

Today's ISPs generate motion information internally and discard it after the frame is processed. Instead, we keep the motion information from each frame and expose it at the SoC-level to inmprove the efficiency of the vision backend.

2.3. Motion Estimation using Block Matching

Among various motion estimation algorithms, block-matching (BM) [74] is widely used in ISP algorithms due to its balance between accuracy and efficiency. Here, we briefly introduce its algorithms and data structures that we will refer to later.



(a) Block-matching. (b) Motion vectors.

Fig. 3: Motion estimation. (a): Block-matching example in a $(2d+1)\times(2d+1)$ search window. L is the macroblock size. (b): Each arrow reprensents an MB's motion vector. MBs in the foreground object have much more prounced motions than the background MBs.

The key idea of BM is to divide a frame into multiple $L \times L$ macroblocks (MB), and search in the previous frame for the closest match for each MB using Sum of Absolute Differences (SAD) of all L^2 pixels as the matching metric. The search is performed within a 2-D search window with $(2d+1)$ pixels in both vertical and horizontal directions, where d is the search range. Fig. 3a illustrates the basic concepts.

Different BM strategies trade-off search accuracy with compute efficiency. The most accurate approach is to perform an exhaustive search (ES) within a search window, which requires $L^2 \cdot (2d+1)^2$ arithmetic operations per MB. Other BM variants trade a small accuracy loss for computation reduction. For instance, the classic three step search (TSS) [79] searches only part of the search window by decreasing d in logarithmic steps. TSS simplifies the amount of arithmetic operations per MB to $L^2 \cdot (1+8 \cdot log_2(d+1))$, which is a 8/9 reduction under $d=7$. We refer interested readers to Jakubowski and Pastuszak [74] for a comprehensive discussion of BM algorithms.

Eventually, BM calculates a *motion vector* (MV) for each MB, which represents the location offset between the MB and its closest match in the previous frame as illustrated in Fig. 3a. Critically, this offset can be used as an estimation of the MB's motion. For instance, an MV $<u,v>$ for an MB at location $<x,y>$ indicates that the MB is moved from location $<x+u, y+v>$ in the previous frame. Fig. 3b visualizes the motion vectors in a frame. Note that MVs can be encoded efficiently. An MV requires $\lceil log_2(2d+1) \rceil$ bits for each direction, which equates to just 1 byte of storage under a typical d of seven.

3. Motion-based Continuous Vision Algorithm

Pixel changes across frames in a continuous video stream directly encode object motion. Thus, pixel-level temporal motion information can be used to simplify continuous vision tasks through motion extrapolation. This section first provides an overview of the algorithm (Sec. 3.1). We then discuss two important aspects of the algorithm: how to extrapolate (Sec. 3.2) and when to extrapolate (Sec. 3.3).

3.1. Overview

Euphrates makes a distinction between two frame types: Inference frame (I-frame) and Extrapolation frame (E-frame). An I-frame refers to a frame where vision computation such as detection and tracking is executed using expensive CNN inference with the frame pixel data as input. In contrast, an E-frame refers to a frame where visual results are generated by extrapolating ROIs from the previous frame, which itself could either be an I-frame or an E-frame. Fig. 4 illustrates this process using object tracking as an example. Rectangles in the figure represent the ROIs of a single tracked object. Frames at t_0 and t_2 are I-frames with ROIs generated by full CNN inference. On the other hand, ROIs in frames at t_1, t_3, and t_4 are extrapolated from their corresponding preceding frames.

Intuitively, increasing the ratio of E-frames to I-frames reduces the number of costly CNN inferences, thereby enabling higher frame rates while improving energy-efficiency. However, this strategy must have little accuracy impact to be useful. As such, the challenge of our algorithm is to strike a balance between accuracy and efficiency. We identify two aspects that affect the accuracy-efficiency trade-off: *how* to extrapolate from previous frame, and *when* to perform extrapolation.

3.2. How to Extrapolate

The goal of extrapolation is to estimate the ROI(s) for the current frame without CNN inference by using the motion vectors generated by the ISP. Our hypothesis is that the average motion of all pixels in a visual field can largely estimate the field's global motion. Thus, the first step in the algorithm calculates the average motion vector (μ) for a given ROI according to Equ. 1, where N denotes the total number of pixels bounded by the ROI, and $\overrightarrow{v_i}$ denotes the motion vector of the i^{th} bounded pixel. It is important to note that the ISP generates MVs at a macroblock-granularity, and as such each pixel inherits the MV from the MB it belongs to. Sec. 6.3 will show that the MV granularity has little impact on accuracy.

$$\mu = \sum_i^N \overrightarrow{v_i} \; / \; N \tag{1}$$

$$\alpha_F^i = 1 - \frac{SAD_F^i}{255 \times L^2} \tag{2}$$

$$MV_F = \beta \cdot \mu_F + (1-\beta) \cdot MV_{F-1} \tag{3}$$

Extrapolating purely based on average motion, however, has two downsides: it is vulnerable to motion vector noise and it does not consider object deformation.

Filtering Noisy Motion Vectors The block-based motion estimation algorithm introduces noise in the MVs. For instance, when a visual object is blurred or occluded (hidden), the block-matching algorithm may not be able to find a good match within the search window.

To quantify the noise in an MV, we associate a confidence value with each MV. We empirically find that this confidence is highly correlated with the SAD value, which is generated during block-matching. Intuitively, a higher SAD value indicates a lower confidence, and vice versa. Equ. 2 formulates the confidence calculation, where SAD_F^i denotes the SAD value of the i^{th} macroblock in frame F, and L denotes the dimension of the macroblocks. Effectively, we normalize an MV's SAD to the maximum possible value (i.e., $255 \times L^2$) and regulate the resultant confidence (α_F^i) to fall between $[0, 1]$. We then derive the confidence for an ROI by averaging the confidences of all the MVs encapsulated by the ROI.

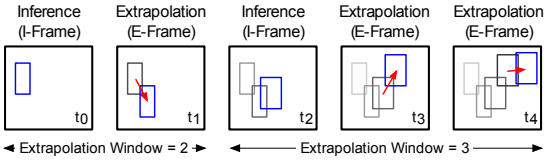


Fig. 4: Vision computation results such as Region of Interest (ROIs) are generated using CNN inference in I-frames. ROIs in E-frames are extrapolated from the previous frame using motion information.

The confidence value can be used to filter out the impact of very noisy MVs associated with a given ROI. This is achieved by assigning a high weight to the average MV in the current frame (μ_F), if its confidence is high. Otherwise, the motion from previous frames is emphasized. In a recursive fashion, this is achieved by directly weighting the contribution of the current MV against the average Equ. 3, where MV_F denotes the final motion vector for frame F, MV_{F-1} denotes the motion vector for the previous frame, and β is the filter coefficient that is determined by α. Empirically, it is sufficient to use a simple piece-wise function that sets β to α if α is greater than a threshold, and to 0.5 otherwise. The final motion vector (MV_F) is composed with the ROI in the previous frame to update its new location. That is: $R_F = R_{F-1} + MV_F$.

Handle Deformations Using one global average motion essentially treats an entire scene as a rigid object, which, however, ignores non-rigid deformations. For instance, the head and the arms of a running athlete have different motions that must be taken into account. Inspired by the classic deformable parts model [65], we divide an ROI into multiple sub-ROIs and apply extrapolation using the above scheme (i.e., Equ. 1 - Equ. 3) for each sub-ROI. In this way, we allow each sub-ROI to move in different directions with different magnitudes. As a result, we get several disconnected sub-ROIs. We then derive the final ROI by calculating the minimal bounding box that encapsulates all the extrapolated sub-ROIs.

Computation Characteristics Our algorithm is very efficient to compute. Consider a typical ROI of size 100×50, the extrapolation step requires only about 10 K 4-bit fixed-point operations per frame, several orders of magnitude fewer than the billions of operations required by CNN inferences.

3.3. When to Extrapolate

Another important aspect of the extrapolation algorithm is to decide which frames to execute CNN inference on, and

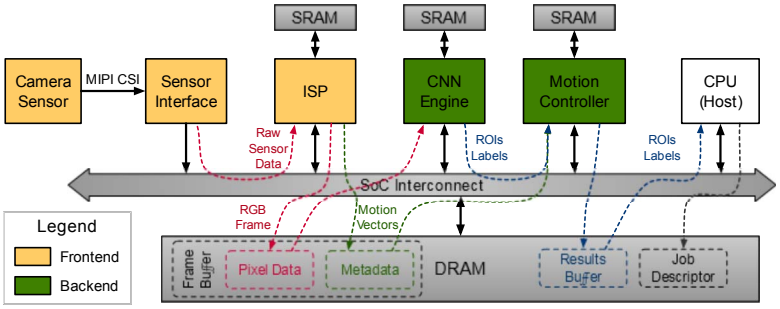


Fig. 5: Block diagram of the augmented continuous vision subsystem in a mobile SoC.

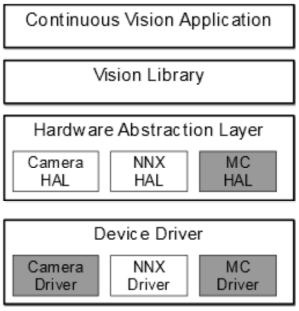


Fig. 6: Vision software stack with modifications shaded.

which frames to extrapolate. To simplify the discussion, we introduce the notion of an Extrapolation Window (EW), which is the number of consecutive frames between two I-frames (exclusive) as shown in Fig. 4. Intuitively, as EW increases, the compute efficiency improves, but errors introduced by extrapolation also start accumulating, and vice versa. Therefore, EW is an important knob that determines the trade-off between compute efficiency and accuracy. Euphrates provides two modes for setting EW: constant mode and adaptive mode.

Constant mode sets EW statically to a fixed value. This provides predictable, bounded performance and energy-efficiency improvements. For instance, under $EW = 2$, one can estimate that the amount of computation per frame is reduced by half, translating to $2\times$ performance increase or 50% energy savings.

However, the constant mode can not adapt to extrapolation inaccuracies. For instance, when an object partially enters the frame, the block-matching algorithm will either not be able to find a good match within its search window or find a numerically-matched MB, which, however, does not represent the actual motion. In such case, CNN inference can provide a more accurate vision computation result.

We introduce a dynamic control mechanism to respond to inaccuracies introduced by motion extrapolation. Specifically, whenever a CNN inference is triggered, we compare its results with those obtained from extrapolation. If the difference is larger than a threshold, we incrementally reduce EW; similarly, if the difference is consistently lower than the threshold across several CNN invocations, we incrementally increase EW.

The two modes are effective in different scenarios. The constant mode is useful when facing a hard energy or frame rate bound. When free of such constraints, adaptive mode improves compute efficiency with little accuracy loss.

4. Architecture Support

In this section, we start from a state-of-the-art mobile SoC, and show how to co-design the SoC architecture with the proposed algorithm. After explaining our design philosophy and providing an overview (Sec. 4.1), we describe the hardware augmentations required in the frontend (Sec. 4.2) and backend of the vision subsystem (Sec. 4.3). Finally, we discuss the software implications of our architecture extensions (Sec. 4.4).

4.1. Design Philosophy and System Overview

Design Principles Two principles guide our SoC design. First, the vision pipeline in the SoC must act autonomously, to avoid constantly interrupting the CPU which needlessly burns CPU cycles and power (Sec. 2.1). This design principle motivates us to provide SoC architecture support for the new algorithm, rather than implementing it in software, because the latter would involve the CPU in every frame.

Second, the architectural support for the extrapolation functionality should be decoupled from CNN inference. This design principle motivates us to propose a separate IP to support the new functionality rather than augmenting an existing CNN accelerator. The rationale is that CNN accelerators are still evolving rapidly with new models and architectures constantly emerging. Tightly coupling our algorithm with any particular CNN accelerator is inflexible in the long term. Our partitioning accommodates future changes in inference algorithm, hardware IP (accelerator, GPU, etc.), or even IP vendor.

System Overview Fig. 5 illustrates the augmented mobile SoC architecture. In particular, we propose two architectural extensions. First, motivated by the synergy between the various motion-enabled imaging algorithms in the ISP and our motion extrapolation CV algorithm, we augment the ISP to expose the motion vectors to the vision backend. Second, to coordinate the backend under the new algorithm without significant CPU intervention, we propose a new hardware IP called the motion controller. The frontend and backend communicate through the system interconnect and DRAM.

Our proposed system works in the following way. The CPU initially configures the IPs in the vision pipeline, and initiates a vision task by writing a job descriptor. The camera sensor module captures real-time raw images, which are fed into the ISP. The ISP generates, for each frame, both pixel data and metadata that are transferred to an allocated frame buffer in DRAM. The motion vectors and the corresponding confidence data are packed as part of the metadata in the frame buffer.

The motion controller sequences operations in the backend and coordinates with the CNN engine. It directs the CNN engine to read image pixel data to execute an inference pass for each I-frame. The inference results, such as predicted

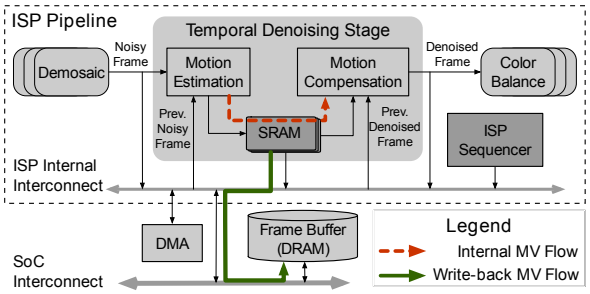


Fig. 7: Euphrates augments the ISP to expose the motion vectors to the rest of the SoC with lightweight hardware extensions. The motion vector traffic is off the critical path and has little performance impact.

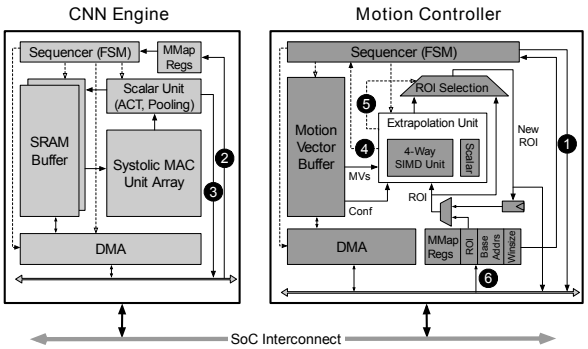


Fig. 8: Euphrates adds the motion controller to the vision backend, alongside an existing, unmodified CNN inference accelerator. Dash lines are control signals and solid lines represent the data flow.

ROIs and possibly classification labels for detected objects, are written to dedicated memory mapped registers in the motion controller through the system interconnect. The motion controller combines the CNN inference data and the motion vector data to extrapolate the results for E-frames.

4.2. Augmenting the Vision Frontend

Motion vectors (MVs) are usually reused internally within an ISP and are discarded after relevant frames are produced. Euphrates exposes MVs to the vision backend, which reduces the overall computation and simplifies the hardware design.

To simplify the discussion, we assume that the motion vectors are generated by the temporal denoising (TD) stage in an ISP. Fig. 7 illustrates how the existing ISP pipeline is augmented. The motion estimation block in the TD stage calculates the MVs and buffers them in a small local SRAM. The motion compensation block then uses the MVs to denoise the current frame. After the current frame is temporally-denoised, the corresponding SRAM space can be recycled. In augmenting the ISP pipeline to expose the MV data, we must decide: 1) by what means the MVs are exposed to the system with minimal design cost, and 2) how to minimize the performance impact on the ISP pipeline.

Piggybacking the Frame Buffer We propose to expose the MVs by storing them in the metadata section of the frame buffer, which resides in the DRAM and is accessed by other SoC IPs through the existing system memory management unit. This augmentation is implemented by modifying the ISP's sequencer to properly configure the DMA engine.

Piggybacking the existing frame buffer mechanism rather than adding a dedicated link between the ISP and the vision backend has the minimum design cost with negligible memory traffic overhead. Specifically, a 1080p frame (1920 $\times$ 1080) with a 16 $\times$ 16 macroblock size will produce 8,100 motion vectors, equivalent to only about 8 KB per frame (Recall from Sec. 2.3 that each motion vector can be encoded in one byte), which is a very small fraction of the 6 MB frame pixel data that is already committed to the frame buffer.

Taking MV Traffic off the Critical-path A naive design to handle MV write-back might reuse the existing local SRAM in the TD stage as the DMA buffer. However, this strategy would stall the ISP pipeline due to SRAM resource contention. This is because the ISP's local SRAMs are typically sized to precisely for the data storage required, thanks to the deterministic data-flow in imaging algorithms. Instead, to take the MV write traffic off the critical path, we propose to double-buffer the SRAM in the TD stage at a slight cost in area overhead. In this way, the DMA engine opportunistically initiates the MV write-back traffic as it sees fit, effectively overlapping the write-back with the rest of the ISP pipeline.

4.3. Augmenting the Vision Backend

We augment the vision backend with a new IP called the motion controller. It's job is two-fold. First, it executes the motion extrapolation algorithm. Second, it coordinates the CNN engine without interrupting the CPU. The CNN accelerator is left unmodified, with the same interface to the SoC interconnect.

We design the motion controller engine as a microcontroller (μC) based IP, similar to many sensor co-processors such as Apple's Motion Co-processor [3]. It sits on the system interconnect, alongside the CNN accelerator. The main difference between our IP and a conventional μC such as ARM Cortex-M series [7] is that the latter typically does not have on-chip caches. Instead, our extrapolation engine is equipped with an on-chip SRAM that stores the motion vectors and is fed by a DMA engine. In addition, the IP replaces the conventional instruction fetch and decode mechanisms with a programmable sequencer, which reduces energy and area, while still providing programmability to control the datapath.

Fig. 8 shows the microarchitecture of the extrapolation controller. Important data and control flows in the figure are numbered. The motion controller is assigned the master role and the CNN engine acts as a slave in the system. The master IP controls the slave IP by using its sequencer to program the slave's memory-mapped registers (❶ and ❷ in Fig. 8). The slave IP always returns the computation results to the master IP (❸) instead of directly interacting with the CPU. We choose this master-slave separation, instead of the other way around,

because it allows us to implement all the control logics such as adaptive EW completely in the extrapolator engine without making assumptions about the CNN accelerator's internals.

The core of the motion controller's datapath is an extrapolation unit which includes a SIMD unit and a scalar unit. The extrapolation operation is highly parallel (Sec. 3.2), making SIMD a nature fit. The scalar unit is primarily responsible for generating two signals: one that controls the EW size in the adaptive mode (❹) and the other that chooses between inferenced and extrapolated results (❺). The IP also has a set of memory-mapped registers that are programmed by the CPU initially and receive CNN engine's inference results (❻).

4.4. Software Implications

Euphrates makes no changes to application developers' programming interface and the CV libraries. This enables software backward compatibility. Modifications to the Hardware Abstraction Layer (HAL) and device drivers are required.

We show the computer vision software stack in Fig. 6 with enhancements shaded. The new motion controller (MC) needs to be supported at both the HAL and driver layer. In addition, the camera driver is enhanced to configure the base address of motion vectors. However, the camera HAL is left intact because motion vectors need not be visible to OS and programmers. We note that the CNN engine (NNX) driver and HAL are also unmodified because of our design decision to assign the master role to the motion controller IP.

5. Implementation and Experimental Setup

This section introduces our hardware modeling methodology (Sec. 5.1) and software infrastructure (Sec. 5.2).

5.1. Hardware Setup

We develop an in-house simulator with a methodology similar to the GemDroid [51] SoC simulator. Our simulator includes a functional model, a performance model, and an power model for evaluating the continuous vision pipeline. The functional model takes in video streams to mimic real-time camera capture and implements the extrapolation algorithm in OpenCV, from which we derive accuracy results. The performance model captures the timing behaviors of various vision pipeline components including the camera sensor, the ISP, the CNN accelerator, and the motion controller. It then models the timing of cross-IP activities, from which we tabulate SoC events that are fed into the power model for energy estimation.

Whenever possible, we calibrate the power model by measuring the Nvidia Jetson TX2 module [19], which is widely used in mobile vision systems. TX2 exposes several SoC and board level power rails to a Texas Instruments INA 3221 voltage monitor IC, from which we retrieve power consumptions through the I2C interface. Specifically, the ISP power is obtained from taking the differential power at the VDD_SYS_SOC power rail between idle and active mode, and the main memory power is obtained through the VDD_SYS_DDR power rail. We develop RTL models or refer to public data sheets when direct measurement is unavailable. Below we discuss how major components are modeled.

Table 1: Details about the modeled vision SoC.

Component	Specification
Camera Sensor	ON Semi AR1335, 1080p @ 60 FPS
ISP	768 MHz, 1080p @ 60 FPS
NN Accelerator (NNX)	24×24 systolic MAC array 1.5 MB double-buffered local SRAM 3-channel, 128bit AXI4 DMA Engine
Motion Controller (MC)	4-wide SIMD datapath 8KB local SRAM buffer 3-channel, 128bit AXI4 DMA Engine
DRAM	4-channel LPDDR3, 25.6 GB/s peak BW

Image Sensor We model the AR1335 [6], an image sensor used in many mobile vision systems including the Nvidia Jetson TX1/TX2 modules [15]. In our evaluation, we primarily consider the common sensing setting that captures 1920 $\times$ 1080 (1080p) videos at 60 FPS. AR1335 is estimated to consume 180 mW of power under this setting [6], representative of common digital camera sensors available on the market [5, 23, 25]. We directly integrate the power consumptions reported in the data sheet into our modeling infrastructure.

ISP We base our ISP modeling on the specifications of the ISP carried on the Jetson TX2 board. We do not model the ISP's detailed microarchitecture but capture enough information about its memory traffic. We make this modeling decision because our lightweight ISP modification has little effect on the ISP datapath, but does impact memory traffic (Sec. 4.2). The simulator generates ISP memory traces given a particular resolution and frame rate. The memory traces are then interfaced with the DRAM simulator as described later.

We take direct measurement of the ISP on TX2 for power estimation. Under 1080p resolution at 60 FPS, the ISP is measured to consume 153 mW, comparable to other industrial ISPs [2]. We could not obtain any public information as to whether the TX2 ISP performs motion estimation. According to Sec. 2.3, a 1080p image requires about 50 million arithmetic operations to generate motion vectors, which is about 2.5% compute overhead compared to a research ISP [70], and will be much smaller compared to commercial ISPs. We thus conservatively factor in 2.5% additional power.

Neural Network Accelerator We develop a systolic array-based CNN accelerator and integrate it into our evaluation infrastructure. The accelerator architecture is reminiscent of the Google Tensor Processing Unit (TPU) [77], but is much smaller, as befits the mobile budget [95].

The accelerator consists of a 24×24 fully-pipelined Multiply-Accumulate array clocked at 1GHz, representing a raw peak throughput of 1.152 TOPS. A unified, double-

Table 2: Summary of benchmarks. GOPS for each neural network is estimated under the 60 FPS requirement. Note that our baseline CNN accelerator provides 1.15 TOPS peak compute capability.

Application Domain	Neural Network	GOPS	Benchmark	Total Frames
Object Detection	Tiny YOLO YOLOv2	675 3423	In-house Video Sequences	7,264
Object Tracking	MDNet	635	OTB 100 VOT 2014	59,040 10,213

buffered SRAM array holds both weights and activations and is 1.5 MB in total. A scalar unit is used to handle per-activation tasks such as scaling, normalization, activation, and pooling.

We implement the accelerator in RTL and synthesize, place, and route the design using Synposys and Cadence tools in a 16nm process technology. Post-layout results show an area of 1.58 mm^2 and a power consumption of 651 mW. This is equivalent to a power-efficiency of 1.77 TOPS/W, commensurate with recent mobile-class CNN accelerators that demonstrate around 1~3 TOPs/W [41, 42, 49, 58, 89, 103] in silicon.

To facilitate future research, we open-source our cycle-accurate simulator of the systolic array-based DNN accelerator, which can provide performance, power and area requirements for a parameterized accelerator on a given DNN model [100].

Motion Controller The motion controller has a light compute requirement, and is implemented as a 4-wide SIMD datapath with 8 KB local data SRAM. The SRAM is sized to hold the motion vectors for one 1080p frame with a 16×16 MB size. The IP is clocked at 100 MHz to support 10 ROIs per frame at 60 FPS, sufficient to cover the peak case in our datasets. We implement the motion controller IP in RTL, and use the same tool chain and 16nm process technology. The power consumption is 2.2 mW, which is just slightly more than a typical micro-controller that has SIMD support (e.g., ARM M4 [12]). The area is negligible (35,000 um^2).

DRAM We use DRAMPower [14] for power estimation. We model the specification of an 8GB memory with 128-bit interface, similar to the one used on the Nvidia Jetson TX2. We further validate the simulated DRAM power consumption against the hardware measurement obtained from the Jetson TX2 board. Under the 1080p and 60 FPS camera capture setting, the DRAM consumes about 230 mW.

5.2. Software Setup

We evaluate Euphrates under two popular mobile continuous vision scenarios: object detection and object tracking. Their corresponding workload setup is summarized in Table 2.

Object Detection Scenario In our evaluation, we study a state-of-the-art object detection CNN called YOLOv2 [97, 98], which achieves the best accuracy and performance among all the object detectors. As Table 2 shows, YOLOv2 requires over 3.4 TOPS compute capability at 60 FPS, significantly exceeding the mobile compute budget. For comparison purposes, we also evaluate a scaled-down version of YOLOv2 called Tiny YOLO. At the cost of 20% accuracy loss [38], Tiny YOLO reduces the compute requirement to 675 GOPS, which is within the capability of our CNN accelerator.

We evaluate object detection using an in-house video dataset. We could not use public object detection benchmarks (such as Pascal VOC 2007 [32]) because they are mostly composed of standalone images without temporal correlation as in real-time video streams. Instead, we capture a series of videos and extract image sequences. Each image is then manually annotated with bounding boxes and labels. The types of object classes are similar to the ones in Pascal VOC 2007 dataset. Overall, each frame contains about 6 objects and the whole dataset includes 7,264 frames, similar to the scale of Pascal VOC 2007 dataset. We plan to release the dataset in the future.

We use the standard Intersect-over-Union (IoU) score as an accuracy metric for object detection [64, 83]. IoU is the ratio between the intersection and the union area between the predicted ROI and the ground truth. A detection is regarded as a true positive (TP) if the IoU value is above a certain threshold; otherwise it is regarded as a false positive (FP). The final detection accuracy is evaluated as $TP/(TP+FP)$ across all detections in all frames, also known as the average precision (AP) score in object detection literature [64].

Visual Tracking Scenario We evaluate a state-of-the-art, CNN-based tracker called MDNet [91], which is the winner of the Video Object Tracking (VOT) challenge [36]. Table 2 shows that compared to object detection, tracking is less compute-intensive and can achieve 60 FPS using our CNN accelerator. However, many visual tracking scenarios such as autonomous drones and video surveillance do not have active cooling. Thus, there is an increasing need to reduce the power/energy consumption of visual tracking [37].

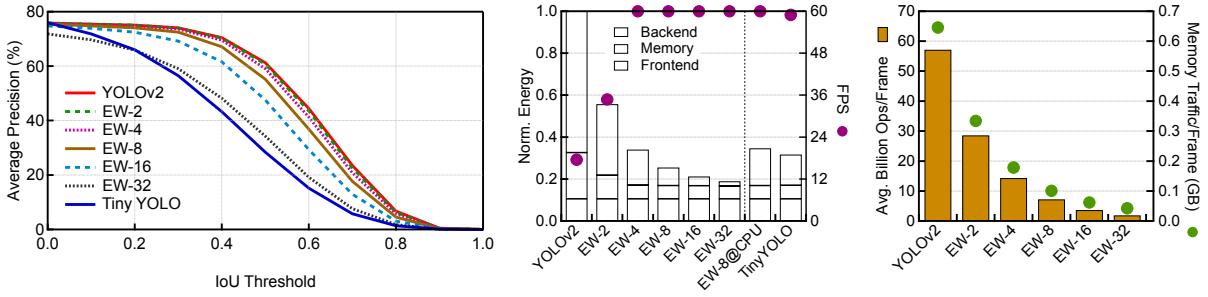
We evaluate two widely used object tracking benchmarks: Object Tracking Benchmark (OTB) 100 [35, 112] and VOT 2014 [36, 39]. OTB 100 contains 100 videos with different visual attributes such as illumination variation and occlusion that mimic realistic tracking scenarios in the wild. VOT 2014 contains 25 sequences with irregular bounding boxes, complementing the OTB 100 dataset. In total, we evaluate about 70,000 frames. We use the standard success rate as the accuracy metric [111], which represents the percentage of detections that have an IoU ratio above a certain threshold.

6. Evaluation

We first quantify the effectiveness of Euphrates under object detection (Sec. 6.1) and tracking (Sec. 6.2) scenarios. We then show that Euphrates is robust against motion estimation results produced in the vision frontend (Sec. 6.3).

6.1. Object Detection Results

Euphrates doubles the achieved FPS with 45% energy saving at the cost of only 0.58% accuracy loss. Compared to the conventional approach of reducing the CNN compute cost by



(a) Average precision comparison. (b) Energy and FPS comparison. (c) Compute and memory comparison.

Fig. 9: Average precision, normalized energy consumption, and FPS comparisons between various object detection schemes. Energy is broken-down into three main components: backend (CNN engine and motion controller), main memory, and frontend (sensor and ISP).

scaling down the network size, Euphrates achieves a higher frame rate, lower energy consumption, and higher accuracy.

Accuracy Results Fig. 9a compares the average precision (AP) between baseline YOLOv2 and Euphrates under different extrapolation window sizes (EW-N, where N ranges from 2 to 32 in powers of 2). For a comprehensive comparison, we vary the IoU ratio from 0 (no overlap) to 1 (perfect overlap). Each <x, y> point corresponds to the percentage of detections (y) that are above a given IoU ratio (x). Overall, the AP declines as the IoU ratio increases.

Replacing expensive NN inference with cheap motion extrapolation has negligible accuracy loss. EW-2 and EW-4 both achieve a success rate close to the baseline YOLOv2, represented by the close proximity of their corresponding curves in Fig. 9a. Specifically, under an IoU of 0.5, which is commonly regarded as an acceptable detection threshold [63], EW-2 loses only 0.58% accuracy compared to the baseline YOLOv2.

Energy and Performance The energy savings and FPS improvements are significant. Fig. 9b shows the energy consumptions of different mechanisms normalized to the baseline YOLOv2. We also overlay the FPS results on the right y-axis. The energy consumption is split into three parts: frontend (sensor and ISP), main memory, and backend (CNN engine and motion controller). The vision frontend is configured to produce frames at a constant 60 FPS in this experiment. Thus, the frontend energy is the same across different schemes.

The baseline YOLOv2 consumes the highest energy and can only achieve about 17 FPS, which is far from real-time. As we increase EW, the total energy consumption drops and the FPS improves. Specifically, EW-2 reduces the total energy consumption by 45% and improves the frame rate from 17 to 35; EW-4 reduces the energy by 66% and achieves real-time frame rate at 60 FPS. The frame rate caps at EW-4, limited by the frontend. Extrapolating beyond eight consecutive frames have higher accuracy loss with only marginal energy improvements. This is because as EW size increases the energy consumption becomes dominated by the vision frontend and memory.

The significant energy efficiency and performance improvements come from two sources: relaxing the compute in the backend and reducing the SoC memory traffic. Fig. 9c shows the amount of arithmetic operations and SoC-level memory traffic (both reads and writes) per frame under various Euphrates settings. As EW increases, more expensive CNN inferences are replaced with cheap extrapolations (Sec. 3.2), resulting in significant energy savings. Euphrates also reduces the amount of SoC memory traffic. This is because E-frames access only the motion vector data, and thus avoid the huge memory traffic induced by executing the CNNs (SRAM spills). Specifically, each I-frame incurs 646 MB memory traffic whereas E-frames require only 22.8 MB.

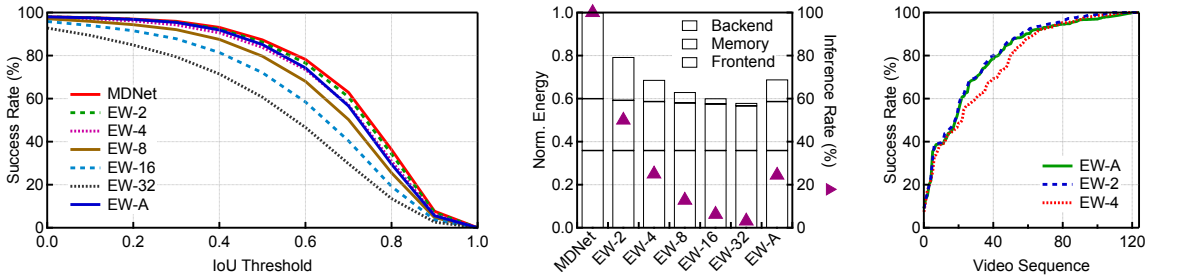
Finally, the second to last column in Fig. 9b shows the total energy of EW-8 when extrapolation is performed on CPU. EW-8 with CPU-based extrapolation consumes almost as high energy as EW-4, essentially negating the benefits of extrapolation. This confirms that our architecture choice of using a dedicated motion controller IP to achieve task autonomy is important to realizing the full benefits in the vision pipeline.

Tiny YOLO Comparison One common way of reducing energy consumption and improving FPS is to reduce the CNN model complexity. For instance, Tiny YOLO uses only nine of YOLOv2's 24 convolutional layers, and thus has an 80% MAC operations reduction (Table 2).

However, we find that exploiting the temporal motion information is a more effective approach to improve object detection efficiency than simply truncating a complex network. The bottom curve in Fig. 9a shows the average precision of Tiny YOLO. Although Tiny YOLO executes 20% of YOLOv2's MAC operations, its accuracy is even lower than EW-32, whose computation requirement is only 3.2% of YOLOv2. Meanwhile, Tiny YOLO consumes about 1.5 $\times$ energy at a lower FPS compared to EW-32 as shown in Fig. 9b.

6.2. Visual Tracking Results

Visual tracking is a simpler task than object detection. As shown in Table 2, the baseline MDNet is able to achieve real-time (60 FPS) frame rate using our CNN accelerator. This section shows that without degrading the 60 FPS frame rate, Euphrates reduces energy consumption by 21% for the vision



(a) Success rate comparison.

(b) Normalized energy consumption and inference rate comparison.

(c) Success rate for all 125 video sequences under an IoU ratio of 0.5.

Fig. 10: Accuracy loss and energy saving comparisons between baseline MDNet and various Euphrates configurations for OTB 100 and VOT 2014 datasets. Energy is dissected into three parts: backend (CNN engine and motion controller), main memory, and frontend (sensor and ISP).

subsystem (50% for the backend), with an 1% accuracy loss.

Results Fig. 10a compares the success rate of baseline MDNet, Euphrates under different EW sizes (EW-N, where N ranges from 2 to 32 in power of 2 steps), and Euphrates under the adaptive mode (EW-A). Similar to object detection, reducing the CNN inference rate using motion extrapolation incurs only a small accuracy loss for visual tracking. Specifically, under an IoU of 0.5 EW-2's accuracy degrades by only 1%.

The energy saving of Euphrates is notable. Fig. 10b shows the energy breakdown under various Euphrates configurations normalized to the baseline MDNet. As a reference, the right y-axis shows the inference rate, i.e., the percentage of frames where a CNN inference is triggered. EW-2 and EW-4 trigger CNN inference for 50% and 25% of the frames, thereby achieving 21% and 31% energy saving compared to MDNet that has an 100% inference rate. The energy savings are smaller than in the object detection scenario. This is because MDNet is simpler than YOLOv2 (Table 2) and therefore, the vision backend contributes less to the overall energy consumption.

Finally, Euphrates provides an energy-accuracy trade-off through tuning EW. For instance, EW-32 trades 42% energy savings with about 27% accuracy loss at 0.5 IoU. Extrapolating further beyond 32 frames, however, would have only marginal energy savings due to the increasing dominance of the vision frontend and memory, as is evident in Fig. 10b.

Adaptive Mode Compared to the baseline MDNet, the adaptive mode of Euphrates (EW-A) reduces energy by 31% with a small accuracy loss of 2%, similar to EW-4. However, we find that the adaptive mode has a more uniform success rate across different tracking scenes compared to EW-4. Fig. 10c shows the success rate of all 125 video sequences in the OTB 100 and VOT 2014 datasets under EW-A, EW-2, and EW-4, sorted from low to high. Each video sequence presents a different scene that varies in terms of tracked objects (e.g., person, car) and visual attributes (e.g., occlusion, blur). EW-A has a higher success rate than EW-4 across most of the scenes, indicating its wider applicability to different object detection scenarios. Compared to EW-2, the adaptive mode has a similar accuracy behavior, but consumes less energy.

6.3. Motion Estimation Sensitivity Study

Euphrates leverages motion vectors generated by the ISP. This section shows that the results of Euphrates are robust against different motion vector characteristics. In particular, we focus on two key characteristics: *granularity* and *quality*. Due to the space limit we only show the results of object tracking on the OTB 100 dataset, but the conclusion holds generally.

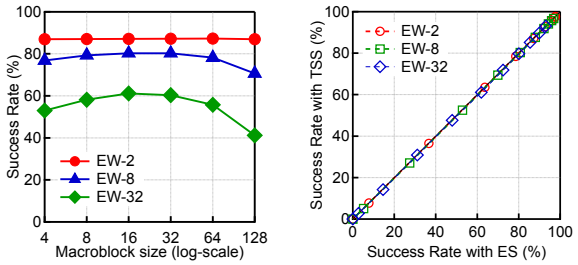
Granularity Sensitivity Motion vector granularity is captured by the macroblock (MB) size during block-matching. Specifically, a $L \times L$ MB represents the motions of all its L^2 pixels using one single vector. Fig. 11a shows how the success rate under an IoU ratio of 0.5 changes with the MB size across three extrapolation windows (2, 8, and 32).

We make two observations. First, Euphrates' accuracy is largely insensitive to the MB size when the extrapolation window is small. For instance, the success rates across different MB sizes are almost identical in EW-2. As the extrapolation window grows from 2 to 32, the impact of motion vector granularity becomes more pronounced. This is because errors due to imprecise motion estimation tend to accumulate across frames under a large extrapolation window.

Second, MBs that are too small (e.g., 4) or too large (e.g., 128) have a negative impact on accuracy (Fig. 11a). This is because overly-small MBs do not capture the global motion of an object, especially objects that have deformable parts such as a running athlete; overly-large MBs tend to mistake a still background with the motion of a foreground object. 16×16 strikes the balance and consistently achieves the highest success rate under large extrapolation windows. It is thus a generally preferable motion vector granularity.

Quality Sensitivity Motion vector quality can be captured by the Sum of Absolute Differences (SAD) between the source and destination macroblocks. Lower SAD indicates higher quality. Different block-matching algorithms trade-off quality with computation cost. As discussed in Sec. 2.3, the most accurate approach, exhaustive search (ES), leads to $9\times$ more operations than the simple TSS algorithm [79].

We find that ES improves the accuracy only marginally



(a) Sensitivity of success rate to different macroblock sizes.

(b) Success rate comparison between ES and TSS.

Fig. 11: Accuracy sensitivity to motion estimation results.

over TSS. Fig. 11b is a scatter plot comparing the success rates between ES (x-axis) and TSS (y-axis) where each marker corresponds to a different IoU ratio. Across three different extrapolation window sizes (2, 8, and 32), the success rates of ES and TSS are almost identical.

7. Discussion

Motion Estimation Improvements Euphrates is least effective when dealing with scenes with fast moving and blurred objects. Let us elaborate using the OTB 100 dataset, in which every video sequence is annotated with a set of visual attributes [35] such as illumination variation. Fig. 12 compares the average precision between MDNet and EW-2 across different visual attribute categories. Euphrates introduces the most accuracy loss in the Fast Motion and Motion Blur category.

Fast moving objects are challenging due to the limit of the search window size of block matching (Sec. 2.3), in which an accurate match is fundamentally unobtainable if an object moves beyond the search window. Enlarging the search window might improve the accuracy, but has significant overhead. Blurring is also challenging because block-matching might return a macroblock that is the best match for a blurred object but does not represent the actual motion.

In the short term, we expect that fast and blurred motion can be greatly alleviated by higher frame rate (e.g., 240 FPS) cameras that are already available on consumer mobile devices [1]. A high frame rate camera reduces the amount of motion variance in consecutive frames and also has very short exposure time that diminishes motion blur [66]. In the long term, we believe that it is critical to incorporate non-vision sensors such as an Inertial Measurement Unit [22] as alternative sources for motion [3, 80], and combine vision vs. non-vision sensors for accurate motion estimation, as exemplified in the video stabilization feature in the Google Pixel 2 smartphone [11].

Hardware Design Alternatives The video codec is known for using block-matching algorithms for video compression [67]. However, video codecs are trigged only if real-time camera captures are to be stored as video files. This, however, is not the dominant use case in continuous vision where camera captures are consumed as sensor inputs by the computer vision algorithms, rather than by humans.

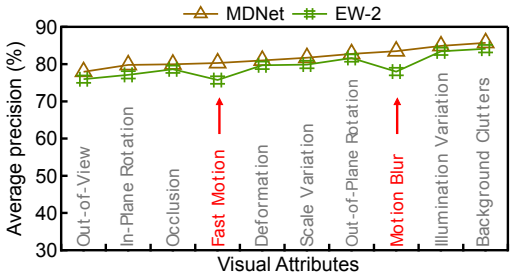


Fig. 12: Accuracy sensivitity to different visual attributes.

That being said, video codecs complement ISPs in scenarios where ISPs are not naturally a part of the image processing pipeline. For instance, if video frames are streamed from the Internet, which are typically compressed, or are retrieved from compressed video files, video codecs can readily provide motion vectors to augment the continuous vision tasks whereas ISPs typically are idle during these tasks. We leave it as future work to co-design the codec hardware with the vision backend.

8. Related Work

Motion-based Continuous Vision Euphrates exploits the temporal motion information for efficient continuous vision, an idea recently getting noticed in the computer vision community. Zhang et al. [116] and Chadha et al. [45] propose CNN models for action recognition using motion vectors as training inputs. TSN [107] trains a CNN directly using multiple frames. Our algorithm differs from all above in that Euphrates does not require any extra training effort. In the case of TSN which is also evaluated using OTB 100 datasets, our algorithm ends up achieving about 0.2% higher accuracy.

Fast YOLO [101] reuses detection results from the previous frame if insufficient motion is predicted. Euphrates has two advantages. First, Fast YOLO requires training a separate CNN for motion prediction while Euphrates leverages motion vectors. Second, Fast YOLO does not perform extrapolation due to the lack of motion estimation whereas Euphrates extrapolates using motion vectors to retain high accuracy.

Finally, Euphrates is composable with all the above systems as they can be used as the baseline inference engine in Euphrates. Our algorithm and hardware extensions do not modify the baseline CNN model, but improves its energy-efficiency.

Energy-Efficient Deep Learning Much of the recent research on energy-efficient CNN architecture has focused on designing better accelerators [40, 48, 62, 77, 102], leveraging emerging memory technologies [50, 78, 105], exploiting sparsity and compression [60, 69, 92, 110, 115], and better tooling [76, 86, 96]. Euphrates takes a different but complementary approach. While the goal of designing better CNN architectures is to reduce energy consumption per inference, Euphrates reduces the rate of inference by replacing inferences with simple extrapolation and thus saves energy.

Suleiman et al. [106] quantitatively show that classic CV algorithms based on hand-crafted features are extremely energy-efficient at the cost of large accuracy loss compared to CNNs. Euphrates shows that motion extrapolation is a promising way to bridge the energy gap with little accuracy loss.

Specialized Imaging & Vision Architectures Traditional ISPs perform only primitive image processing while leaving advanced photography and vision tasks such as high dynamic range and motion estimation to CPUs and GPUs. Modern imaging and vision processors are capable of performing advanced tasks *in-situ* in response to the increasing compute requirements of emerging algorithms. For instance, Google's latest flagship smartphone, Pixel 2, has a dedicated SoC equipped with eight Image Processing Units for advanced computational photography algorithms such as HDR+ [31]. IDEAL [87] accelerates the BM3D-based image denoising algorithms. Mazumdar et al. [88] design specialized hardware for face-authentication and stereoscopic video processing.

Researchers have also made significant effort to retain programmability in specialized vision architectures [46, 93, 108]. For instance, Movidius Myriad 2 [41] is a VLIW-based vision processor used in Google Clip camera [17] and DJI Phantom 4 drone [33]. Clemons et al. [53] propose a specialized memory system for imaging and vision processors while exposing a flexible programming interface. Coupled with the programmable architecture substrate, domain specific languages such as Halide [94], Darkroom [70], and Rigel [71] offer developers even higher degree of flexibility to implement new features. We expect imaging and vision processors in future to be highly programmable, offering more opportunities to synergistically architect image processing and continuous vision systems together as we showcased in this paper.

Computer Vision on Raw Sensor Data Diamond et al. [59] and Buckler et al. [44] both showed that CNN models can be effectively trained using raw image sensor data. RedEye [81] and ASP Vision [47] both move early CNN layer(s) into the camera sensor and compute using raw sensor data. This line of work is complementary to Euphrates in that our algorithm makes no assumption about which image format motion vectors are generated. In fact, recent work has shown that motion can be directly estimated from raw image sensor data using block matching [43, 114]. We leave it as future work to port Euphrates to support raw data.

9. Conclusion

Delivering real-time continuous vision in an energy-efficient manner is a tall order for mobile system design. To overcome the energy-efficiency barrier, we must expand the research horizon from individual accelerators toward holistically co-designing different mobile SoC components. This paper demonstrates one such co-designed system to enable motion-based synthesis. It leverages the temporal motion information naturally produced by the imaging engine (ISP) to reduce the compute demand of the vision engine (CNN accelerator).

Looking forward, exploiting the synergies across different SoC IP blocks will become ever more important as mobile SoCs incorporate more specialized domain-specific IPs. Future developments should explore cross-IP information sharing beyond just motion metadata and expand the co-design scope to other on/off-chip components. Our work serves the first step, not the final word, in a promising new direction of research.

References

[1] "A Closer Look at Slow Motion Video on the iPhone6." https://www.wired.com/2015/01/closer-look-slow-motion-video-iphone-6/ 11

[2] "AP0101CS High-Dynamic Range (HDR) Image Signal Processor (ISP)." http://www.onsemi.com/pub/Collateral/AP0101CS-D.PDF 7

[3] "Apple Motion Coprocessors." https://en.wikipedia.org/wiki/Apple_motion_coprocessors 6, 11

[4] "Apple's Neural Engine Infuses the iPhone with AI Smarts." https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/ 2

[5] "AR0542 CMOS Digital Image Sensor." https://www.onsemi.com/pub/Collateral/AR0542-D.PDF 7

[6] "AR1335 CMOS Digital Image Sensor." https://www.framos.com/media/pdf/06/d8/9a/AR1335-D-PDF-framos.pdf 7

[7] "Arm Cortex-M Series Processors." https://developer.arm.com/products/processors/cortex-m 6

[8] "ARM Mali Camera." https://www.arm.com/products/graphics-and-multimedia/mali-camera 3

[9] "Arm Object Detection Processor." https://developer.arm.com/products/processors/machine-learning/arm-od-processor 3

[10] "ASICFPGA Camera Image Signal Processing Core." http://asicfpga.com/site_upgrade/asicfpga/pds/isp_pds_files/ASICFPGA_ISP_Core_v4.0_simple.pdf 3

[11] "Behind the Motion Photos Technology in Pixel 2." https://research.googleblog.com/2018/03/behind-motion-photos-technology-in.html 11

[12] "Cortex-M4 Technical Reference Manual, Revision r0p0." https://static.docs.arm.com/ddi0439/b/DDI0439B_cortex_m4_r0p0_trm.pdf 8

[13] "DENALI-MC HDR ISP." http://pinnacleimagingsystems.com/embedded-products/ 3

[14] "DRAMPower: Open-source DRAM Power & Energy Estimation Tool." https://github.com/tukl-msd/DRAMPower 8

[15] "e-CAM131_CUTX2 - 4K MIPI NVIDIA Jetson TX2/TX1 Camera Board." https://www.e-consystems.com/13mp-nvidia-jetson-tx2-camera-board.asp 7

[16] "Enhancing Augmented Reality with Advanced Object Detection Techniques." https://www.qualcomm.com/invention/research/projects/computer-vision/3d-object-detection 3

[17] "Google's Clips camera is powered by a tailor-made AI chip." https://www.theverge.com/circuitbreaker/2017/10/6/16434834/google-clips-camera-ai-movidius-myriad-vpu 12

[18] "Hikvision Advanced Image Processing: Noise Reduction." http://oversea-download.hikvision.com/UploadFile/file/Hikvision_Advanced_Image_Processing--Noise_Reduction.pdf 3

[19] "Jetson TX2 Module." http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html 7

[20] "MIPI Camera Serial Interface 2 (MIPI CSI-2)." https://www.mipi.org/specifications/csi-2 2

[21] "Movidius Myriad X VPU Product Brief." https://uploads.movidius.com/1503874473-MyriadXVPU_ProductBriefaug25.pdf 1

[22] "MPU-9250 Product Specification Revision 1.1." https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf 11

[23] "MT9P031 CMOS Digital Image Sensor." http://www.onsemi.com/pub/Collateral/MT9P031-D.PDF 7

[24] "Nvidia ADAS." https://www.nvidia.com/en-us/self-driving-cars/adas/ 3

[25] "OmniVision OV5693." http://www.ovt.com/sensors/OV5693 7

[26] "Photoshop CS2 HDR." https://luminous-landscape.com/photoshop-cs2-hdr/ 3

[27] "PX4FLOW Smart Camera." https://pixhawk.org/modules/px4flow 3

[28] "Qualcomm pioneers new active depth sensing module." https://www.qualcomm.com/news/onq/2017/08/15/qualcomm-pioneers-new-active-depth-sensing-module 3

[29] "Second Version of HoloLens HPU will Incorporate AI Coprocessor for Implementing DNNs." https://www.microsoft.com/en-us/research/blog/second-version-hololens-hpu-will-incorporate-ai-coprocessor-implementing-dnns/ 2

[30] "Snapdragon 835 Mobile Platform." https://www.qualcomm.com/products/snapdragon/processors/835 3

[31] "Surprise! The Pixel 2 is hiding a custom Google SoC for image processing." https://arstechnica.com/gadgets/2017/10/the-pixel-2-contains-a-custom-google-soc-the-pixel-visual-core/ 12

[32] "The PASCAL Visual Object Classes Challenge 2007." http://host.robots.ox.ac.uk/pascal/VOC/voc2007/ 1, 8

[33] "The revolutionary chipmaker behind Google's project Tango is now powering DJI's autonomous drone." https://www.theverge.com/2016/3/16/11242578/movidius-myriad-2-chip-computer-vision-dji-phantom-4 12

[34] "Vision Chips." http://www.centeye.com/technology/vision-chips/ 3

[35] "Visual Tracker Benchmark." http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html 8, 11

[36] "VOT2014 Benchmark." http://www.votchallenge.net/vot2014/ 8

[37] "What are the Toughest Challenges in ADAS Design." http://www.electronicdesign.com/power/what-are-toughest-challenges-adas-design 8

[38] "YOLO: Real-Time Object Detection." https://pjreddie.com/darknet/yolo/ 8

[39] "The Visual Object Tracking VOT2014 challenge results," in *Proc. of CVPRW*, 2015. 8

[40] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free Deep Neural Network Computing," in *Proc. of ISCA*, 2016. 11

[41] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on Vision Processing Unit for Mobile Applications," *IEEE Micro*, 2015. 1, 8, 12

[42] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H.-J. Yoo, "14.6 A 0.62 mW Ultra-Low-Power Convolutional-Neural-Network Face-recognition Processor and a CIS Integrated with Always-on Haar-like Face Detector," in *Proc. of ISSCC*, 2017. 8

[43] G. Boracchi and A. Foi, "Multiframe Waw-data Denoising based on Block-matching and 3-D Filtering for Low-light Imaging and Stabilization," in *Proc. Int. Workshop on Local and Non-Local Approx. in Image Processing*, 2008. 12

[44] M. Buckler, S. Jayasuriya, and A. Sampson, "Reconfiguring the Imaging Pipeline for Computer Vision," in *Proc. of ICCV*, 2017. 12

[45] A. Chadha, A. Abbas, and Y. Andreopoulos, "Video Classification With CNNs: Using The Codec As A Spatio-Temporal Activity Sensor," in *arXiv:1710.05112*, 2017. 2, 11

[46] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. Al Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini, "Exploring Architectural Heterogeneity in Intelligent Vision Systems," in *Proc. of HPCA*, 2015. 12

[47] H. G. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. Molnar, "ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks using Angle Sensitive Pixels," in *Proc. of CVPR*, 2016. 12

[48] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks," in *Proc. of ISCA*, 2016. 11

[49] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *Proc. of ISSCC*, 2017. 8

[50] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *Proc. of ISCA*, 2016. 11

[51] N. Chidambaram Nachiappan, P. Yedlapalli, N. Soundararajan, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "GemDroid: A Framework to Evaluate Mobile Platforms," 2014. 7

[52] B.-D. Choi, J.-W. Han, C.-S. Kim, and S.-J. Ko, "Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, 2007. 3

[53] J. Clemons, C.-C. Cheng, I. Frosio, D. Johnson, and S. W. Keckler, "A patch memory system for image processing and computer vision," in *Proc. of MICRO*, 2016. 12

[54] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge university press, 2000. 1

[55] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proc. of CVPR*, 2005. 1

[56] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg, "ECO: Efficient Convolution Operators for Tracking," in *Proc. of CVPR*, 2017. 1

[57] P. E. Debevec and J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," in *Proc. of SIGGRAPH*, 1997. 3

[58] G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh *et al.*, "14.1 A 2.9 TOPS/W Deep Convolutional Neural Network SoC in FD-SOI 28nm for Intelligent Embedded Systems," in *Proc. of ISSCC*, 2017. 8

[59] S. Diamond, V. Sitzmann, S. Boyd, G. Wetzstein, and F. Heide, "Dirty Pixels: Optimizing Image Classification Architectures for Raw Sensor Data," in *Proc. of CVPR*, 2017. 12

[60] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan *et al.*, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-CirculantWeight Matrices," in *Proc. of MICRO*, 2017. 11

[61] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast Feature Pyramids for Object Detection," *PAMI*, 2014. 1

[62] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proc. of ISCA*, 2015. 11

[63] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge: A Retrospective," *IJCV*, 2015. 9

[64] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *IJCV*, 2009. 8

[65] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models," *PAMI*, 2010. 3, 4

[66] H. K. Galoogahi1, A. Fagg, C. Huang, D. Ramanan, and S. Lucey, "Need for Speed: A Benchmark for Higher Frame Rate Object Tracking," in *arXiv:1703.05884*, 2017. 11

[67] M. Ghanbari, *Standard Codecs: Image Compression to Advanced Video Coding*. Institution Electrical Engineers, 2003. 11

[68] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction," in *Proc. of HPCA*, 2016. 2

[69] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proc. of ISCA*, 2016. 11

[70] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan, "Darkroom: Compiling High-Level Image Processing Code into Hardware Pipelines," in *Proc. of SIGGRAPH*, 2014. 7, 12

[71] J. Hegarty, R. Daly, Z. DeVito, J. Ragan-Kelley, M. Horowitz, and P. Hanrahan, "Rigel: Flexible Multi-Rate Image Processing Hardware," in *Proc. of SIGGRAPH*, 2016. 12

[72] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed Tracking with Kernelized Correlation Filters," *PAMI*, 2015. 3

[73] Y. Inoue, T. Ono, and K. Inoue, "Adaptive frame-rate optimization for energy-efficient object tracking," in *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016, p. 158. 2

[74] M. Jakubowski and G. Pastuszak, "Block-based Motion Estimation Algorithms–A Survey," *Opto-Electronics Review*, 2013. 3

[75] H. Ji, C. Liu, Z. Shen, and Y. Xu, "Robust Video Denoising using Low Rank Matrix Completion," in *Proc. of CVPR*, 2010. 3

[76] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "NEUTRAMS: Neural Network Transformation and Co-design Under Neuromorphic Hardware Constraints," in *Proc. of MICRO*, 2016. 11

[77] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, R. C. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proc. of ISCA*, 2017. 7, 11

[78] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *Proc. of ISCA*, 2016. 11

[79] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion Compensated Interframe Coding for Video Conferencing," in *Proc. Nat. Telecommunications Conf.*, 1981. 3, 10

[80] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, "Head tracking for the Oculus Rift," in *Proc. of ICRA*, 2014. 11

[81] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision," in *Proc. of ISCA*, 2016. 12

[82] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our Glass: An Energy and Heat Characterization of Google Glass," in *Proc. of APSys*, 2014. 2

[83] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Proc. of ECCV*, 2014. 8

[84] C. Liu and W. T. Freeman, "A High-Quality Video Denoising Algorithm based on Reliable Motion Estimation," in *Proc. of ECCV*, 2010. 3

[85] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *Proc. of ECCV*, 2016. 1

[86] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning," in *Proc. of HPCA*, 2016. 11

[87] M. Mahmoud, B. Zheng, A. D. Lascorz, F. Heide, J. Assouline, P. Boucher, E. Onzon, and A. Moshovos, "IDEAL: Image DEnoising AcceLerator," in *Proc. of MICRO*, 2017. 12

[88] A. Mazumdar, T. Moreau, S. Kim, M. Cowan, A. Alaghi, L. Ceze, M. Oskin, and V. Sathe, "Exploring Computation-Communication Tradeoffs in Camera Systems," in *Proc. of IISWC*, 2017. 12

[89] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W Subword-parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI," in *Proc. of ISSCC*, 2017. 8

[90] N. C. Nachiappan, H. Zhang, J. Ryoo, N. Soundararajan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das, "VIP: Virtualizing IP Chains on Handheld Platforms," in *Proc. of ISCA*, 2015. 2

[91] H. Nam and B. Han, "Learning Multi-Domain Convolutional Neural Networks for Visual Tracking," in *Proc. of CVPR*, 2016. 1, 3, 8

[92] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," in *Proc. of ISCA*, 2017. 11

[93] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing," in *Proc. of ISCA*, 2013. 12

[94] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines," in *Proc. of PLDI*, 2013. 12

[95] B. Reagen, R. Adolf, P. Whatmough, G.-Y. Wei, and D. Brooks, *Deep Learning for Computer Architects*. Morgan & Claypool Publishers, 2017. 7

[96] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," in *Proc. of ISCA*, 2016. 11

[97] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. of CVPR*, 2016. 1, 3, 8

[98] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *arXiv:1612.08242*, 2016. 1, 8

[99] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks," in *Proc. of NIPS*, 2015. 1

[100] A. Samajdar, Y. Zhu, and P. N. Whatmough, "Scale-sim," https://github.com/ARM-software/SCALE-Sim, 2018. 8

[101] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," in *arXiv:1709.05943v1*, 2017. 2, 11

[102] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," 2016. 11

[103] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "14.6 a 1.42 TOPS/W Deep Convolutional Neural Network Recognition Processor for Intelligent IoE Systems," in *Proc. of ISSCC*, 2016. 8

[104] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. of ICLR*, 2014. 1

[105] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Proc. of HPCA*, 2017. 11

[106] A. Suleiman, Y.-H. Chen, J. Emer, and V. Sze, "Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision," in *Proc. of ISCAS*, 2017. 12

[107] Z. Teng, J. Xing, Q. Wang, C. Lang, S. Feng, and Y. Jin, "Robust Object Tracking based on Temporal and Spatial Deep Networks," in *Proc. of ICCV*, 2017. 11

[108] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Evaluating Programmable Architectures for Imaging and Vision Applications," in *Proc. of MICRO*, 2016. 12

[109] P. Viola and M. J. Jones, "Robust Real-time Object Detection," *IJCV*, 2004. 1

[110] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G. Y. Wei, "14.3 a 28nm soc with a 1.2ghz 568nj/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for iot applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 242–243. 11

[111] Y. Wu, J. Lim, and M.-H. Yang, "Online Object Tracking: A Benchmark," in *Proc. of CVPR*, 2013. 8

[112] ——, "Object Tracking Benchmark," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. 8

[113] J. Yan, Z. Lei, L. Wen, and S. Z. Li, "The Fastest Deformable Part Model for Object Detection," in *Proc. of CVPR*, 2014. 1

[114] C.-C. Yang, S.-M. Guo, and J. S.-H. Tsai, "Evolutionary Fuzzy Block-Matching-Based Camera Raw Image Denoising," in *IEEE Transactions on Cybernetics*, 2017. 12

[115] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism," in *Proc. of ISCA*, 2017. 11

[116] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang, "Real-time Action Recognition with Enhanced Motion Vector CNNs," in *Proc. of CVPR*, 2016. 2, 11

[117] Y. Zhu, M. Mattina, and P. Whatmough, "Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective," *ArXiv e-prints*, Jan. 2018. 1