Live Video Analytics with FPGA-based Smart Cameras

Shang Wang[†‡], Chen Zhang[†], Yuanchao Shu[†], Yunxin Liu[†]
Microsoft Research[†], University of Electronic Science and Technology of China[‡]

ABSTRACT

Analyzing video feeds from large camera networks requires enormous compute and bandwidth. Edge computing has been proposed to ease the burden by bringing resources to the proximity of data. However, the number of cameras keeps growing and the associated computing resources on edge will again fall in short. To fundamentally solve the resource scarcity problem and make edge-based live video analytics scalable, we present an FPGA-based smart camera design that enables efficient in-situ streaming processing to meet the stringent low-power, energy-efficient, low-latency requirements of edge vision applications. By leveraging FPGA's intrinsic properties of architecture efficiency and exploiting its hardware support for parallelism, we demonstrate a 49x speedup over CPU and 6.4x more energy-efficiency than GPU, verified using a background subtraction algorithm.

CCS CONCEPTS

• **Computing methodologies** → *Computer vision problems*; • **Computer systems organization** → *Distributed architectures*; • **Hardware** → *Hardware accelerators.*

KEYWORDS

Edge computing; Cameras; Video analytics; FPGA; DNN

ACM Reference Format:
Shang Wang, Chen Zhang, Yuanchao Shu, Yunxin Liu. 2019. Live Video Analytics with FPGA-based Smart Cameras. In *2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo '19), October 21, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3349614.3356027

1 INTRODUCTION

We are witnessing a huge increase in the number of surveillance cameras in recent years. UK, for example, has one security camera for every 14 people, and the total number of cameras in London alone will go up to one million by 2025 [1]. Analyzing video feeds from the ever-growing camera network poses huge system challenges, among which the cost of compute and network are major ones.

To ease the burden of live video analytics, edge computing has been proposed to bring resources to the proximity of data. However, edge server could still be bottlenecked by limited compute resources as the number of associated cameras increases. To fundamentally solve the resource scarcity problem and make edge-based live video analytics scalable, we argue that cameras should be designed smart to be able to filter out a large portion of video content and only send those necessary content to the edge server for computation-intensive processing.

In this paper, we present an FPGA-based smart-camera design that enables efficient streaming processing to meet the unique characteristics of video surveillance applications. Our design sits upon FPGA, not only utilizing its intrinsic properties of low latency and energy efficiency, but also exploiting its hardware support for parallelism and its unique capability of reconfiguration. Compared to general-purpose processors, FPGA can efficiently avoid system and memory-copy overheads. Its hard-wired circuit design is also critical for low-latency image processing. Compared to ASIC designs, FPGA provides high flexibility to be re-configured to any functionality. In addition, for applications that do not fully occupy the on-board resources, FPGA enables flexible trade-off to either allow concurrent execution of multiple small (heterogeneous) tasks, or optimize a single task for low latency or high accuracy by using more on-board resources. Thus, we may jointly optimize the computation resource allocation and task scheduling for heterogeneous tasks in an energy-efficient manner.

To validate our design, we chose background subtraction as a sample filter and fully implemented it on a state-of-the-art FPGA platform. Our evaluation results show that compared to CPU and GPU based solutions, our method is able to provide the best performance trade-off between processing latency and energy efficiency. Specifically, we achieved 49x speedup on the number of images processed per second and 6.1x better energy-efficiency on Joule per frame over CPU. Compared to using an RTX 2080 GPU, FPGA-based solution is only 48% slower with more than 90% energy reduction.

HotEdgeVideo '19, October 21, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6928-2/19/10…$15.00
https://doi.org/10.1145/3349614.3356027

2 DESIGN OVERVIEW

We adopt a hybrid live video analytics architecture comprised of smart cameras, edge servers and cloud (Figure 1), and primarily focus on video surveillance applications, e.g., traffic counting. Cameras are equipped with FPGA where primitive computer vision (CV) functions like background subtraction are implemented in a streaming fashion. Only filtered frames are then sent to the edge server for further processing such as DNN-based object detection and re-identification. Although different applications may require variant functions, we believe such a split between CV primitives and batch processing tasks (e.g., DNN inference) can be easily implemented while at the same time making full use of the advantages of both FPGA and GPU.

Figure 1: Live video analytics pipeline with FPGA-based smart cameras.

2.1 FPGA Primer

In a nutshell, FPGA [2, 3] is a device composed of an array of re-configurable logic blocks, arithmetic engines (DSPs), block RAM (BRAM), and switchable connection wires. Each logic block is a look-up table (LUT) that can be configured to any combined computation logic. With proper designs and connections, the sea of LUTs and DSPs can collaborate together and act as many parallel "cores" with arbitrary parallelism design and customized functions for performance (e.g., latency and throughput) optimization. BRAM works as parallel random accessed storage on the chip. Typically, a piece of off-the-shelf FPGA contains tens of thousands LUTs, thousands of DSPs and hundreds of Block RAMs [4].

2.2 FPGA-powered Smart Camera

FPGA's special advantages are appealing for achieving the goals of live video analytics. FPGA is good at latency-sensitive jobs due to the circuit-level customization on its massively parallel computing units and on-chip storage banks, saving a large portion of overheads in general-purpose processors, including instruction fetch, cache miss, task scheduling etc. In addition, FPGA's Multiple Instruction, Multiple Data (MIMD) architecture guarantees high concurrency. With proper circuit partition, it is able to support multiple DNN models running at the same time without any time-multiplexing. FPGA is naturally energy-efficient, making it friendly for small form factor cameras.

GPU is one of the most popular acceleration hardware because of its massively parallel architecture. However, FPGA fits much better for streaming applications for two reasons. First, GPU is optimized as an accessory accelerator that are plugged in system buses like PICe. Calling GPUs usually requires system interactions and multiple memory copies between CPU and GPU, which introduces a lot of additional overhead. However, with the hardwired connection, FPGA has the ability to stream data in and out directly, which eliminates both of those system overheads and memory copies. Second, the energy consumption is crucial to battery-powered mobile systems. The gap in energy consumption makes FPGA more attractive for power-hungry continuous mobile vision.

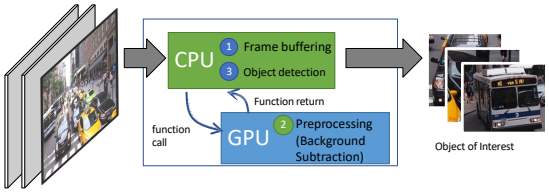

Figure 2: Smart camera with GPU.

Figure 2 shows a typical system architecture design of a smart camera with a GPU accelerator connected to a CPU through a bus connection. As can be seen, CPU is the central of this system. It first fetches videos from the sensor and arises GPU acceleration for image processing. After GPU function call returns, CPU finalizes pre-processing. Although several mobile GPUs share memory with CPU and thus has less overhead, interactions between CPU and GPU persist in current programming models. To reduce those overheads, we propose a bump-in-wire architecture design as shown in Figure 3. Our design lets FPGA directly get video stream from the camera sensor and feed pre-processed images to CPU, thus minimizing system overheads.

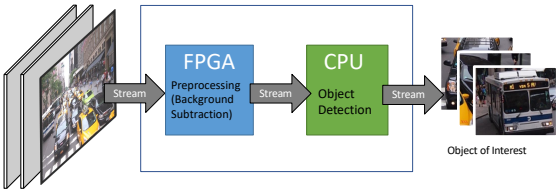

Figure 3: Our smart camera design with FPGA.

However, FPGA suffers from two drawbacks. First, compared to general-purpose processors, programming FPGA uses hardware specific language and usually incurs a longer developing and compilation cycle. Hence, we leverage high-level synthesis, which uses C-based automated design process that interprets an algorithmic description of a desired

behavior and creates digital hardware that implements that behavior. Second, configuring FPGA takes longer time than launching executables on CPUs or GPUs. We propose to configure the most frequently used functions on FPGA and will only re-configure FPGA when necessary.

3 BACKGROUND SUBTRACTION AND FPGA ACCELERATION

In this paper, we use background subtraction, a CV primitive widely used in video analytics tasks, as an example to show how we leverage FPGA to improve the system performance. Below we describe how background subtraction works, followed by our FPGA acceleration design.

3.1 Background Subtraction

The main purpose of background subtraction is to distinguish the moving targets (called foregrounds) of the sequences in the video stream from the background without prior knowledge of the target information. In this paper, we use the Gaussian Mixture Model (GMM) [5], a classic background subtraction algorithm, to detect moving targets. GMM is robust and shows good performance in various conditions such as low illumination and highly compressed videos. The algorithm works on individual pixels and thus there is no across-pixel dependency, making it a good fit for FPGA for parallel processing at pixel level.

Algorithm 1 describes how the GMM algorithm works. The algorithm takes video frames as input and outputs corresponding masks that label each pixel of the frame whether it is background or foreground. For each pixel (x, y) in a frame, the algorithm maintains and filters this pixel with a set of Gaussian Distribution Models (GDMs) that describe the Gaussian probability with three parameters $\langle w, u, \sigma \rangle$, where w, u and σ are the weight, expectation and standard deviation of GDM, respectively. This pixel will be judged as background if it falls in one of the GDMs. Equation 1 describes how the output of a pixel is calculated in a GDM using a threshold λ. $I(x, y, t)$, $u(x, y, t)$, $\sigma(x, y, t)$ and $\sigma^2(x, y, t)$ represent the pixel value, mean, standard deviation and variance at time t and position (x, y) in the video image sequence, respectively. A pixel is background if the difference between its current value at time t and its last mean value at time $t-1$ falls below the multiplication of λ and the most recent standard deviation at time $t-1$. The number of total GDMs is usually empirically pre-defined. We set it to five by default.

$$output = \begin{cases} 0, |I(x,y,t) - u(x,y,t-1)| < \lambda\sigma(x,y,t-1) \\ 1, otherwise \end{cases} \quad (1)$$

Input: Original video frame
Output: Background subtracted frame
Initialization;
while *Next frame* **do**
  while *Next pixel* **do**
    while *Next Gaussian Distribution Model (GDM)* **do**
      if *Match the current GDM* **then**
        Update the w, u and σ of the current GDM;
        Sort GDMs in descending order according to w and σ;
      else
        Reduce the w of the current GDM;
      end
    end
    if *Do not match any GDM* **then**
      if *The number of GDMs reaches the upper limit* **then**
        Remove the last GDM;
      else
        Add a new GDM;
      end
    else
      Pass
    end
    Normalize the w of GDMs;
    Obtain a cutoff index of GDMs based on a customized threshold;
    Determine if the pixel belongs to foreground;
  end
end

Algorithm 1: GMM algorithm.

Before taking in the next pixel, the GMM algorithm updates the current GDM according to Equation 2, where α presents the update frequency which is used to against the small variances in the background such as sun light changes or shaking leaves.

$$\begin{cases} u(x,y,0) & = (1-\alpha)u(x,y,t-1) + \alpha u(x,y,t) \\ \sigma^2(x,y,t) & = (1-\alpha)\sigma^2(x,y,t-1) + \alpha[I(x,y,t) - u(x,y,t)]^2 \\ \sigma(x,y,t) & = \sqrt{\sigma^2(x,y,t)} \end{cases} \quad (2)$$

With the above algorithm, we are able to extract background in a frame and differentiate objects. We use background subtraction as precursor of object detection and region of interest extraction to improve the efficiency of video processing without negatively affecting accuracy. We set a threshold to filter out static frames. When the number of

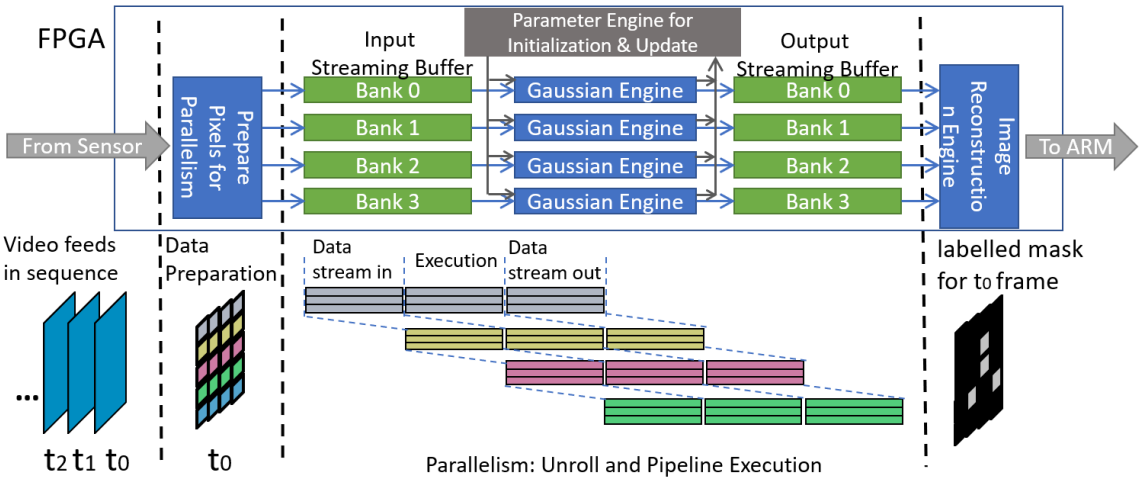

Figure 4: GMM accelerator on FPGA.

pixels of the moving object detected in a video frame exceeds this threshold, we mark the corresponding original frame as a frame of interest, and then pass it to edge server for DNN inference. Otherwise, we mark the corresponding original frame as absence of object of interest and drop it directly.

3.2 FPGA Acceleration

We have designed and implemented the GMM background subtraction algorithm on FPGA. Our accelerator can quickly and efficiently perform the pre-processing operation of GMM background subtraction on the FPGA device, and thus reduce the CPU load and the processing latency.

Figure 4 depicts the detailed design of accelerator engines on FPGA. We apply two levels of parallelism, namely, unrolling and pipelining. Each pixel within a video frame is independent from others and thus can be executed concurrently. However, the dimensions of a frame may range from hundreds to thousands (e.g., 360x480, 1080x1920). It is impossible to fully unroll all dimensions because the number of parallel cores on FPGA is limited and fixed. We thereby cut the input image into multiple slices with a fixed size of pixel batch. Within each pixel batch, we fully unroll the execution so that all Gaussian Engines are running concurrently. When processing the current batch, FPGA accelerator streams in the next pixel batch and streams out the previous pixel batch. The sequences of video frames are executed in such a three-stage pipeline manner. In order to fulfill the execution parallelism, we also customize multi-banked streaming input and output buffers as well as the corresponding data preparing/collecting engine to avoid memory conflicts or caching overheads. As we will show in the next section, such an accelerator is able to achieve high throughput and energy efficiency.

4 EVALUATION

Experimental setup. We ran the GMM algorithm on several different types of hardware, including the ARM Cortex-A53 CPU, Intel Core i7-3770 CPU, NVIDIA RTX 2080 GPU, and Xilinx FPGA using the Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [6]. The dataset used in the experiments is a camera video of a traffic intersection with a resolution of 360x480. Since GMM is composed of multiple element-wise operations, our accelerator can easily scale out to higher resolutions by extending streaming length.

For the experiments on Intel Core i7-3770 CPU and NVIDIA RTX 2080 GPU, we ran the GMM algorithm using OpenCV version 3.4.5 [7] on Ubuntu 16.04 LTS. For the experiments on ARM Cortex-A53 CPU, we used the Xilinx SDx [8] to compile and generate executive file from source code written in C++ with OpenCV. For Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit, we used Xilinx SDx and Vivado HLS to accelerate IP creation by using C++ specifications to be directly targeted into Xilinx programmable devices without manually creating RTL.

Results. Table 2 shows the resource utilization of FPGA given by Vivado HLS. Our GMM implementation uses a large portion of FPGA's hardware resource. This is because, to best optimized the performance, we employed a very deep pipeline so that the accelerator could take new data in on every clock cycle, which costed a lot of registers or Flip-Flop (FF). In future work, we will further study how to utilize other resources for higher parallelism and better balance.

Table 1 shows the performance of the GMM algorithm on different hardware. We evaluate the throughput of each device. We used *img/sec* as the metric as shown in Table 1, which represents the average number of processed frames

Processor	ARM CPU	Intel CPU	GPU	FPGA
Device	Cortex-A53	i7-3770	RTX2080	Zynq UltraScale+
IC tech (nm)	20	22	16	16
Freq (GHz)	1.4	3.40	1.0	0.3
Power (watt)	2.5	146.40	250	20
Img/sec	35.34 (1x)	232.56 (6.6x)	3333.33 (94.3x)	1736.11 (49.13x)
GFLOPS/sec	2.91 (1x)	19.13 (6.6x)	274.17 (94.3x)	142.72 (49.13x)
Joule/img	0.0707 (1x)	0.6295 (0.11x)	0.0750 (0.94x)	0.0116 (6.1x)

Table 1: Performance of GMMs Algorithm on Devices for 360x480 frame resolution.

Resource	DSP	BRAM	LUT	FF
Used	90	8	50,541	382,729
Available	2,520	1,824	274,080	548,160
Utilization	3	≈ 0	18	69

Table 2: FPGA Resource utilization.

per second. The GMMs algorithm on all CPU and GPU platforms were implemented using the OpenCV library. Among all implementations, the beefy RTX 2080 GPU had the best performance, achieving a throughput of up to 3,333.33 *img/sec* at a cost of 250 Watt. The throughput of using FPGA has the second best throughput as high as 1,736.11 *img/sec*. To put it into perspective, FPGA accelerator achieves more than half of GPU performance with only less than 10% of GPU power. In comparison, Intel and ARM CPUs achieve lower throughput, with only 232.56 *img/sec* and 35.34 *img/sec*, respectively. Note that the performance of FPGA accelerator is typically constrained by the overall arithmetic unit and BRAM storage. With larger FPGA, the performance of GMM accelerator could be greatly improved.

Since different types of hardware have different parallel opportunities, it is difficult to directly compare the energy-efficiency between them. In order to provide a fair comparison, we present the results of *Joule/img* in Table 1. It is defined as the average energy (joules) used to process each frame of the video stream, and thus can represent the efficiency of running GMM on different hardware platforms. The power consumption of the RTX 2080 GPU is the highest amount all the devices, as high as 250 Watts. The Intel and ARM CPUs have a power around 146.40 Watts and 2.5 Watts, respectively. As shown in the last row of Table 1, our implementation achieves the highest energy efficiency, which is nearly 6x better than the ARM CPU that is the second best hardware.

5 RELATED WORK

Many systems are designed to use a combination of cameras, edge clusters, and cloud resources to analyze video streams [9–12], including low-cost model design [10, 13, 14], partitioned processing [15–18], adaptation [9, 19, 20] and computation/memory sharing [21–23]. However, these systems focus on video analysis and queries, not video pre-processing. Our work leverages FPGA for efficient video pre-processing and thus is complementary with the existing work on video analytics.

Recently, FPGA has emerged as a promising solution of customized accelerator for many applications, especially for those that need stream processing [24–26]. By customizing dedicated pipelines, concurrent processing units, specified operators, etc., application designers can accelerate many workloads by orders of magnitude using FPGA [27–32]. However, most of previous works are using FPGA as an acceleration card on a bus. Our design directly bumps the FPGA in the wire between front sensors. As far as we know, we are the first attempt of efficient implementation of background subtraction algorithm together with system integration with video analytic applications that detect moving objects in video streams.

6 CONCLUSION AND FUTURE WORK

The increasing popularity of large-scale camera deployments and the advancement of urban intelligence have created enormous challenges for computing capability and network communication bandwidth. In this paper, by deploying the GMM algorithm of background subtraction on the FPGA to filter out the static video frames that do not need to be processed, the burden of the smart camera on the whole system is reduced, and the resource utilization is improved. We believe that using FPGA to perform efficient video processing can stimulate more explorations in future research on edge computing.

In the future, we will integrate FPGA-powered smart cameras with edge servers and cloud to build end-to-end applications. We will further study the system challenges of smart cameras at city-scale, where massive cameras are working together with different workloads and jobs. In addition, FPGA's flexible configurability enables runtime re-configuration and dynamic job scheduling. We will further exploit the chances of software/hardware co-design and running intelligent jobs with higher efficiency and performance.

REFERENCES

[1] How many CCTV Cameras are there in London 2019? https://www.cctv.co.uk/how-many-cctv-cameras-are-there-in-london/, 2019.

[2] Pong P Chu. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version.* John Wiley & Sons, 2011.

[3] Ian Kuon, Russell Tessier, Jonathan Rose, et al. Fpga architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

[4] Pierre Maillard, Michael Hart, Jeff Barton, Praful Jain, and James Karp. Neutron, 64 MeV proton, thermal neutron and alpha single-event upset characterization of Xilinx 20nm UltraScale Kintex FPGA. In *2015 IEEE Radiation Effects Data Workshop*, 2015.

[5] Zoran Zivkovic et al. Improved adaptive gaussian mixture model for background subtraction. In *ICPR (2)*, pages 28–31, 2004.

[6] Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html, 2019. [Online; accessed June-30-2019].

[7] OpenCV Release 3.4.5. https://docs.opencv.org/3.4.5/, 2019. [Online; accessed June-30-2019].

[8] Introduction to the SDx Environments. https://www.xilinx.com/html$_{d}$ocs/xilinx2017$_{4}$/sdaccel$_{d}$oc/fht1504034314910.html, 2019. [Online; accessed June-30-2019].

[9] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable Adaptation of Video Analytics. In *ACM SIGCOMM*, 2018.

[10] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[11] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *ACM MobiSys*, 2018.

[12] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *ACM MobiSys*, 2016.

[13] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *ACM MobiCom*, 2018.

[14] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *IEEE CVPR*, 2017.

[15] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *ACM SenSys*, 2015.

[16] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *IEEE/ACM SEC*, 2018.

[17] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *ACM MobiCom*, 2015.

[18] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. Demo: Video Analytics - Killer App for Edge Computing. In *ACM MobiSys*, 2019.

[19] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *USENIX NSDI*, 2017.

[20] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph E. Gonzalez. Scaling Video Analytics Systems to Large Camera Deployments. In *ACM HotMobile*, 2019.

[21] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *USENIX ATC*, 2018.

[22] Robert LiKamWa and Lin Zhong. Starfish: Efficient concurrency support for computer vision applications. In *ACM MobiSys*, 2015.

[23] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *ACM MobiSys*, 2017.

[24] Roland Dobai and Lukas Sekanina. Image filter evolution on the xilinx zynq platform. In *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, pages 164–171. IEEE, 2013.

[25] Gooitzen Van der Wal, David Zhang, Indu Kandaswamy, Jim Marakowitz, Kevin Kaighn, Joe Zhang, and Sek Chai. FPGA acceleration for feature based processing applications. In *IEEE CVPR*, 2015.

[26] Rafal Kapela, Karol Gugala, Pawel Sniatala, Aleksandra Swietlicka, and Krzysztof Kolanowski. Embedded platform for local image descriptor based object detection. *Applied Mathematics and Computation*, 267:419–426, 2015.

[27] Matthew Russell and Scott Fischaber. Opencv based road sign recognition on zynq. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 596–601. IEEE, 2013.

[28] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *ACM/SIGDA FPGA*, 2015.

[29] TaiLin Han, Guangw Wen Liu, Hua Cai, and Bo Wang. The face detection and location system based on zynq. In *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 835–839. IEEE, 2014.

[30] Yujie Zhang, Meihua Xu, and Huaming Shen. Design of face detection system based on fpga. In *International Conference in Swarm Intelligence*, pages 404–410. Springer, 2013.

[31] Shaowu Pan, Liwei Shi, Shuxiang Guo, Ping Guo, Yanlin He, and Rui Xiao. A low-power soc-based moving target detection system for amphibious spherical robots. In *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1116–1121. IEEE, 2015.

[32] Gorka Velez, Ainhoa Cortés, Marcos Nieto, Igone Vélez, and Oihana Otaegui. A reconfigurable embedded vision system for advanced driver assistance. *Journal of Real-Time Image Processing*, 10(4):725–739, 2015.