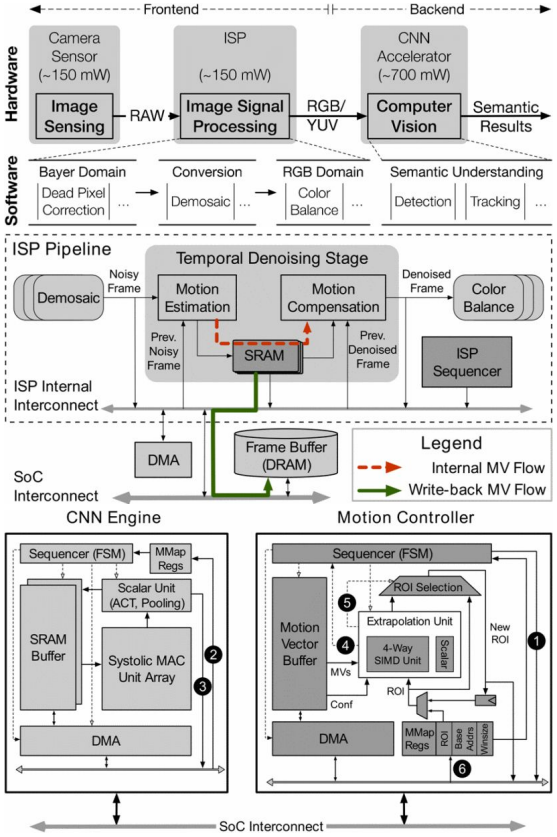


EUPHRATES: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision

Zhu et. al.

Presenters: Enakshi Deb, Haechan Jeong, Manasa Buravalli, Miles Hanbury

EECS 507, F22, University of Michigan

2022.11.10

Section Split

1. Introduction 1p
2. Background and Motivation 1.5p
3. Motion-Based Continuous Vision Algorithm 1.5p
4. Architecture Support 2p
5. Implementation and Experimental Setup 1.5p
6. Evaluation 2p
7. Discussion 0.5p
8. Related Work 0.75p
9. Conclusion 0.25p
10. Suggested Questions / Thoughts

Approx 11p in total -> ~3p per person

Presentation Outline

1. Introduction
2. Background and Motivation
3. Motion-Based Continuous Vision Algorithm
4. Architecture Support
5. Implementation and Experimental Setup
6. Evaluation
7. Discussion
8. Related Work
9. Conclusion
10. Q & A

Acronyms

CNN - Convolutional Neural Networks

SoC - System-on-Chip

TOPS - Tera Operations per second (10^{12})

FPS - Frames per Second

ISP - Image Signal Processor

IP - Intellectual Property

1. Introduction

Introduction

- Computer Vision is the foundation for Advanced Driver Assistance System (ADAS) & Augmented Reality (AR)
- Continuous Computer Vision tasks rely on CNNs to extract semantic information.
- CNNs have replaced handcrafted features such as Haar and HOG, owing to their high accuracy.
- CNNs have compute demands that exceed performance and energy constraints of mobile devices

Introduction

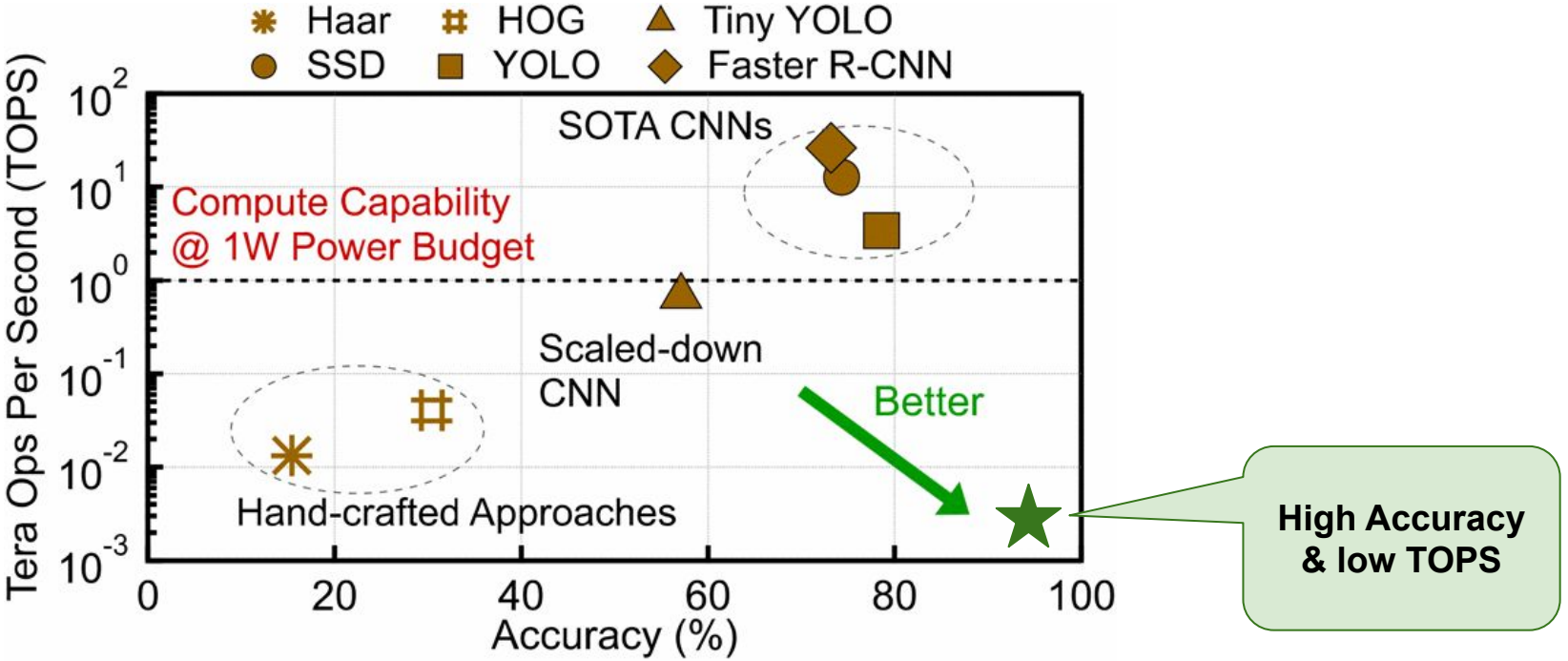


Fig 1: Accuracy and Compute requirement (TOPS) comparison between object detection techniques

Paper Contributions

Goal: improve compute efficiency of continuous vision with small accuracy loss

Key idea: changes in pixel data between consecutive frames represents visual motion.

- First work to exploit sharing motion data across ISP and other IPs
- Propose an algorithm that leverages motion information to reduce number of CNN inferences
- Motion Controller – autonomously coordinates vision pipeline enabling “always-on” vision with very low CPU load

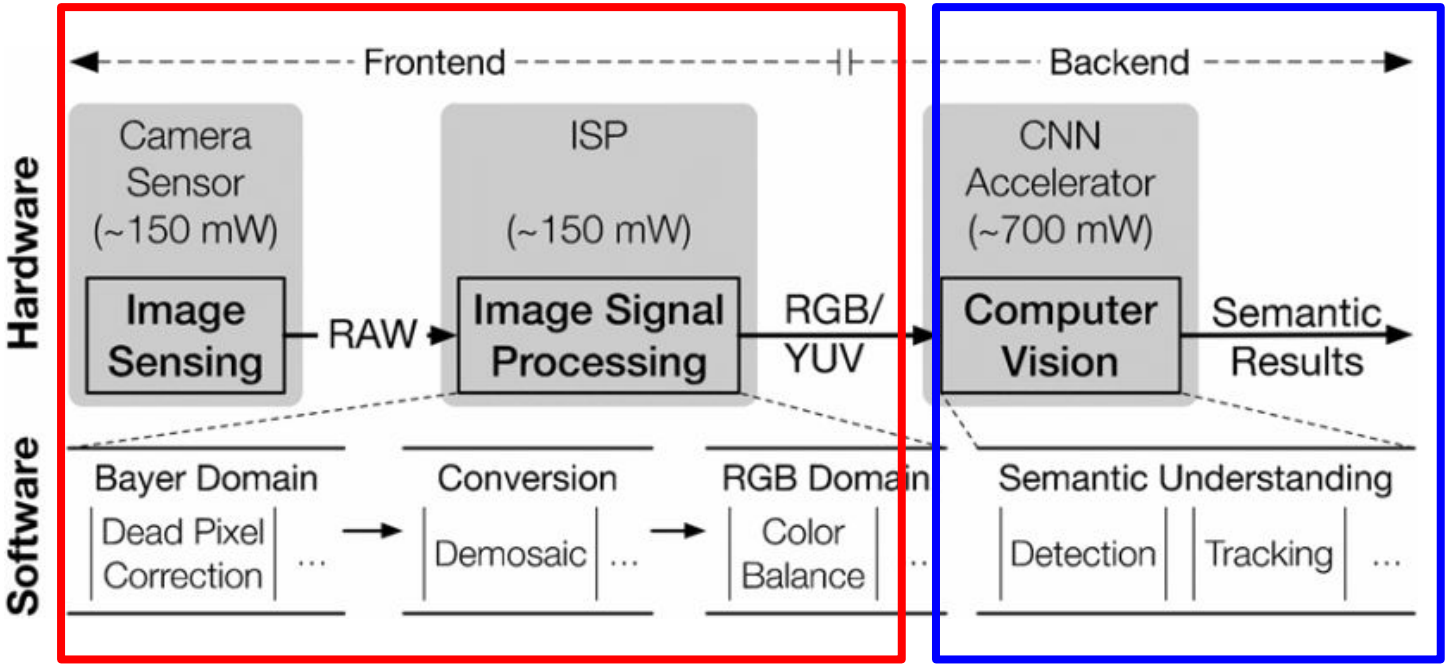
Paper Contributions

- **‘Euphrates’** – a proof-of-concept system of the proposed algorithm-SoC co-design approach
  - Lightweight hardware extension exposing motion information to the vision engine
    - prior work – extract information from offline compressed video or calculate during runtime
  - Achieve energy savings and frame rate improvement
  - Validated with hardware measurements and RTL implementations
  - Evaluated on two tasks – object detection and object tracking

2. Background & Motivation

The Mobile Continuous Vision Pipeline

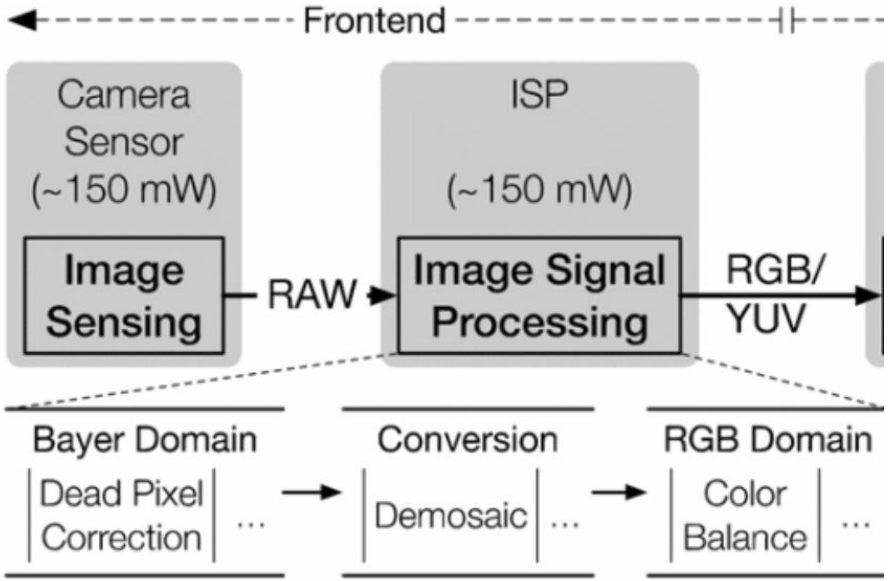
Typical Continuous CV Pipeline



The Mobile Continuous Vision Pipeline

Frontend

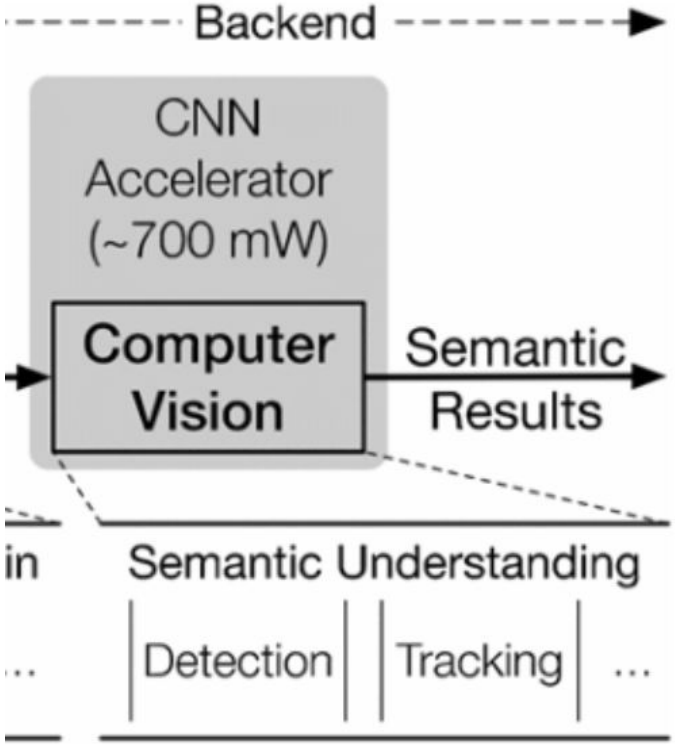
- Prepares pixel data
- Uses camera sensor to capture and produce RAW pixels
- ISP turns RAW data to pixels in RGB/YUV domain



The Mobile Continuous Vision Pipeline

Backend

- Extracts information from images through tasks, i.e., object detection
- CNN accelerator instead of DSP, GPU, and CPU
- Increases task autonomy



The Mobile Continuous Vision Pipeline

Object Tracking

- Localizing a moving object across frames by predicting its region of interest (ROI)

Object Detection

- Simultaneous object classification and localization usually for multiple objects

Motion Estimation in ISPs

- ISPs are integrating photography algorithms that used to be separately performed (e.g. HDR)
- Improves compute efficiency

Motion Estimation

- Estimates how pixels move between consecutive frames
- Used in algorithms i.e. temporal denoising and video stabilization

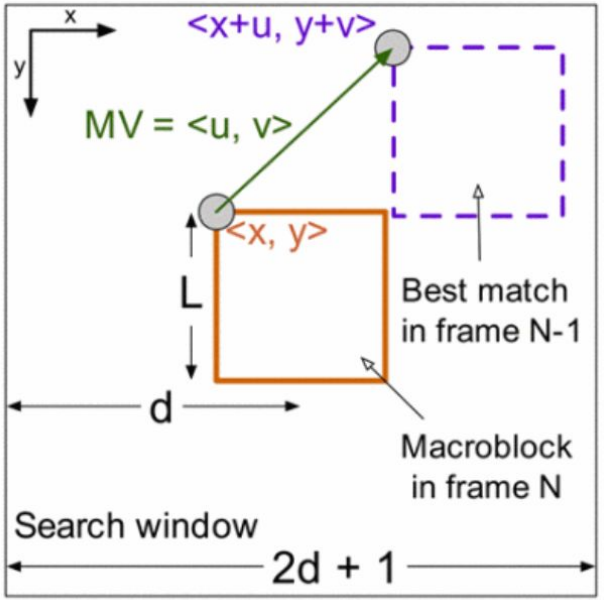
Instead of discarding the motion information after processing, they keep the information and expose it at the SoC-level to improve the vision backend.

Motion Estimation Using Block Matching

Block Matching (BM)

- Divide a frame into multiple L x L MB
- Search in previous frame for closest match for each MB using SAD
- Search performed within 2-D search window
- Trade-off between search accuracy with compute efficiency

MB = macroblock
L = MB size
SAD = sum of absolute differences
MV = motion vector
d = search range

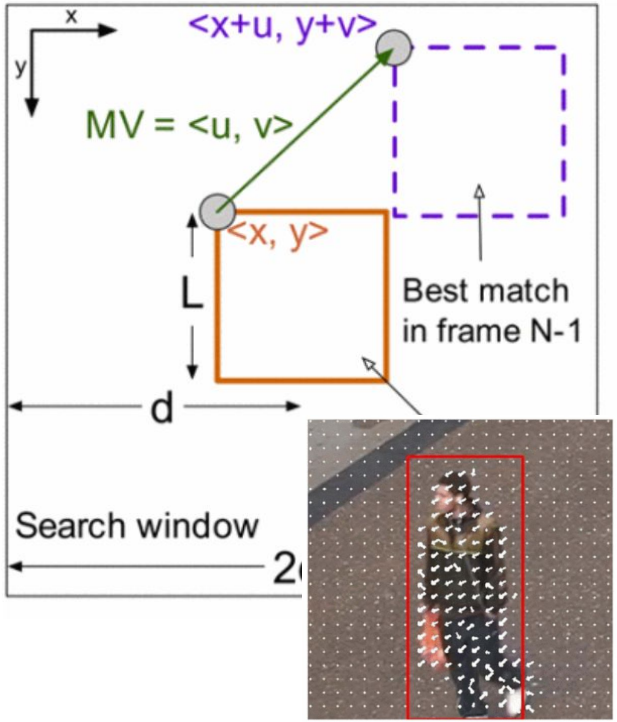


Motion Estimation Using Block Matching

Block Matching (BM)

- BM calculates a MV for each MB
- MV represents the location offset and can be used as motion estimate
- MV <u, v> for an MB <x, y> means that in the previous frame the MB was at location <x+u, y+v>

MB = macroblock
L = MB size
SAD = sum of absolute differences
MV = motion vector
d = search range



3. Motion-Based Continuous Vision Algorithm

Overview

Inference Frame (I-frame)

- Vision computation is executed using expensive CNN inference
- Frame pixel data is input

Extrapolation Frame (E-frame)

- Visual results are generated by extrapolating ROIs from previous frame
- Previous frame could be either type

The challenge of their algorithm is to balance accuracy and efficiency

How to Extrapolate

Algorithm Steps

- Calculate the average MV for a given ROI
  a. Vulnerable to noise and ignores object deformation

MV = motion vector
ROI = region of interest
μ = average MV
$\mathbf{v}_i$ = MV of i-th bounded pixel
N = total # of pixels bounded by the ROI

$$\mu = \sum_i^N \overrightarrow{v_i} \; / \; N \qquad (1)$$

How to Extrapolate

Algorithm Steps

- Quantify the noise in an MV by calculating a confidence value (Eqn 2)
- Correlation exists where higher SAD value indicates a lower confidence
- Compute the ROI confidence by averaging each MV's confidence

MV = motion vector
MB = macroblock
L = MB size
SAD = sum of absolute differences
$\mathbf{SAD}^{i}_{F}$ = SAD value of i^{th} MB in frame F
$\boldsymbol{\alpha}^{i}_{F}$ = confidence

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

How to Extrapolate

Algorithm Steps

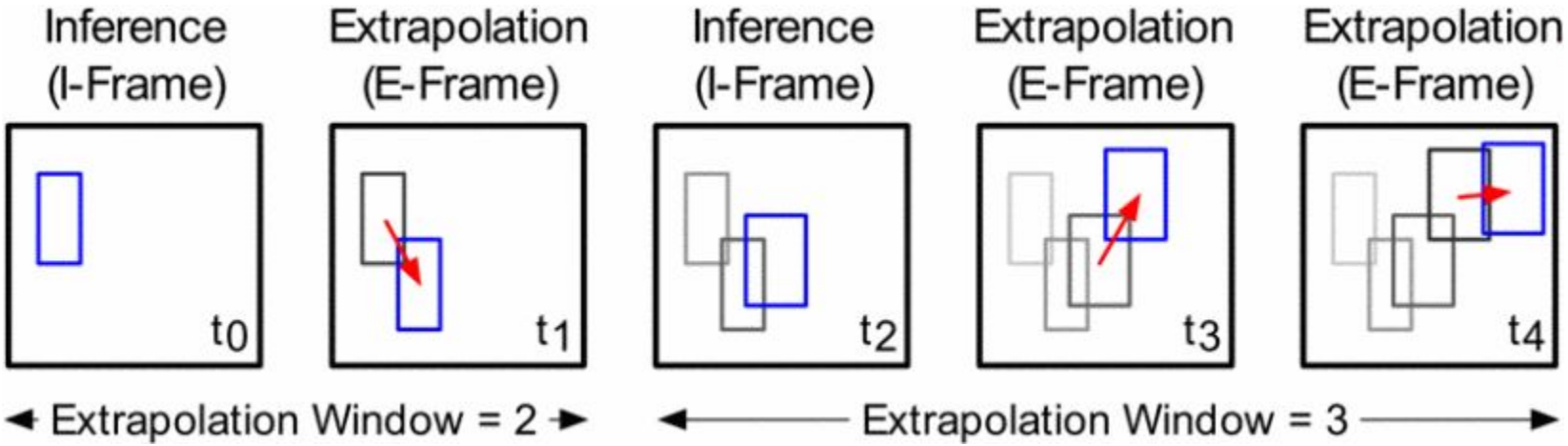
- Filter out the noise (Eqn 3)
- Assign high weight to u_F if confidence is high
- Else motion from previous frame is boosted
- Deformation is handled through sub-dividing ROIs and repeating Eqn 1-3

MV_F = final MV for frame F
MV_{F-1} = previous frame MV
μ_F = average MV
β = filter coefficient
R_F = ROI for frame F
R_{F-1} = previous frame ROI

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

$$R_F = R_{F-1} + MV_F$$

When to Extrapolate



Extrapolation Window (EW)

- # of consecutive frames between two I-frames

When to Extrapolate

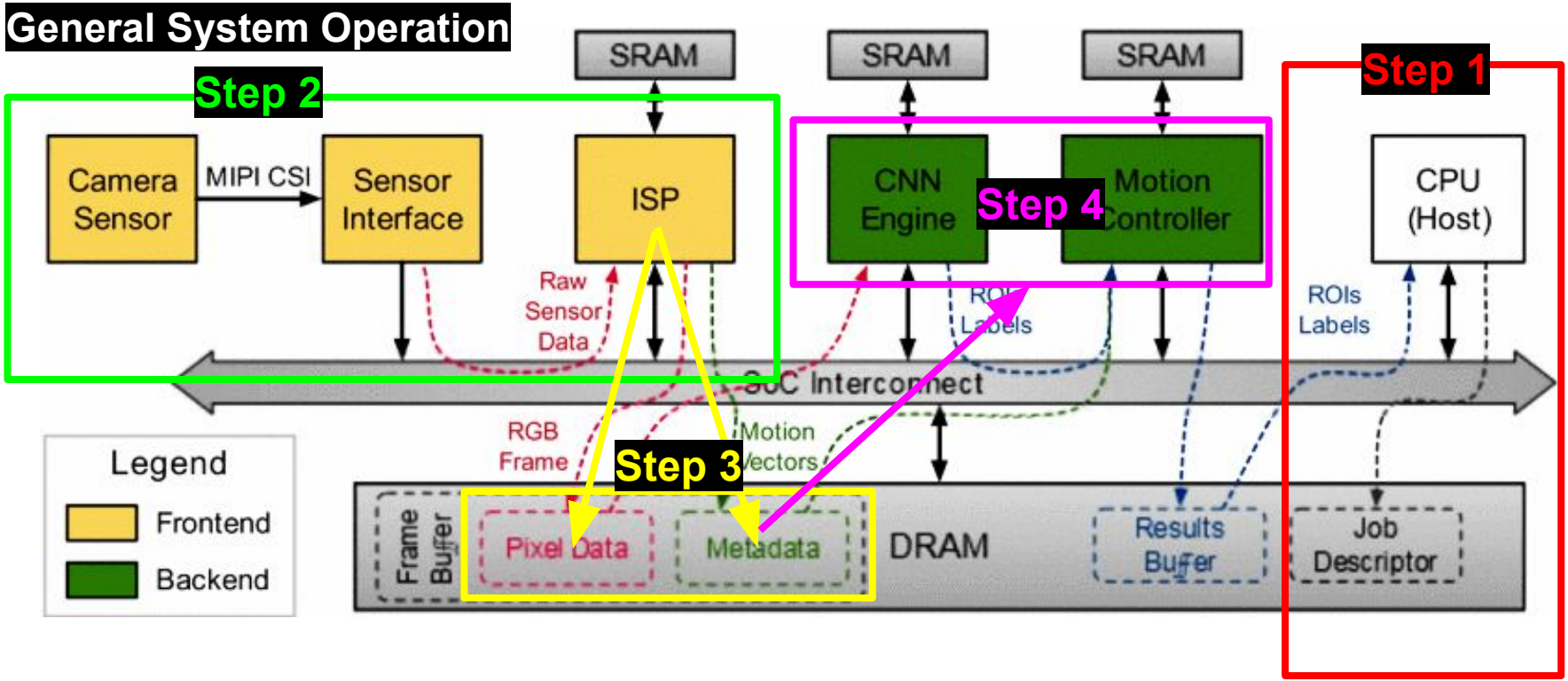
Extrapolation Window (EW)

- Constant Mode
  - EW is set to a fixed value
  - Predictable, but can’t adapt
  - Useful when facing a hard energy or frame rate bound

- Adaptive mode
  - Changes EW value depending on a threshold
  - Comparison between CNN inference and extrapolation results
  - Large difference means reduction in EW

4. Architecture Support

Architecture: Mobile SoC for CV Overview



Architecture: Co-Design Principles

State-of-the-art Mobile SoC → Algorithm Co-designed Mobile SoC

Principle 1. Vision pipeline in the SoC must **avoid interrupting CPU** which needlessly burns CPU cycles and power.

Problem: SW Implementation alone would involve the CPU in every frame.

Solution: Provide **SoC architecture support** for the new algorithm.

Architecture: Co-Design Principles

State-of-the-art Mobile SoC → Algorithm Co-designed Mobile SoC

Principle 1. Vision pipeline in the SoC must **avoid interrupting CPU** which needlessly burns CPU cycles and power.

Problem: SW Implementation alone would involve the CPU in every frame.

Solution: Provide SoC architecture support for the new algorithm.

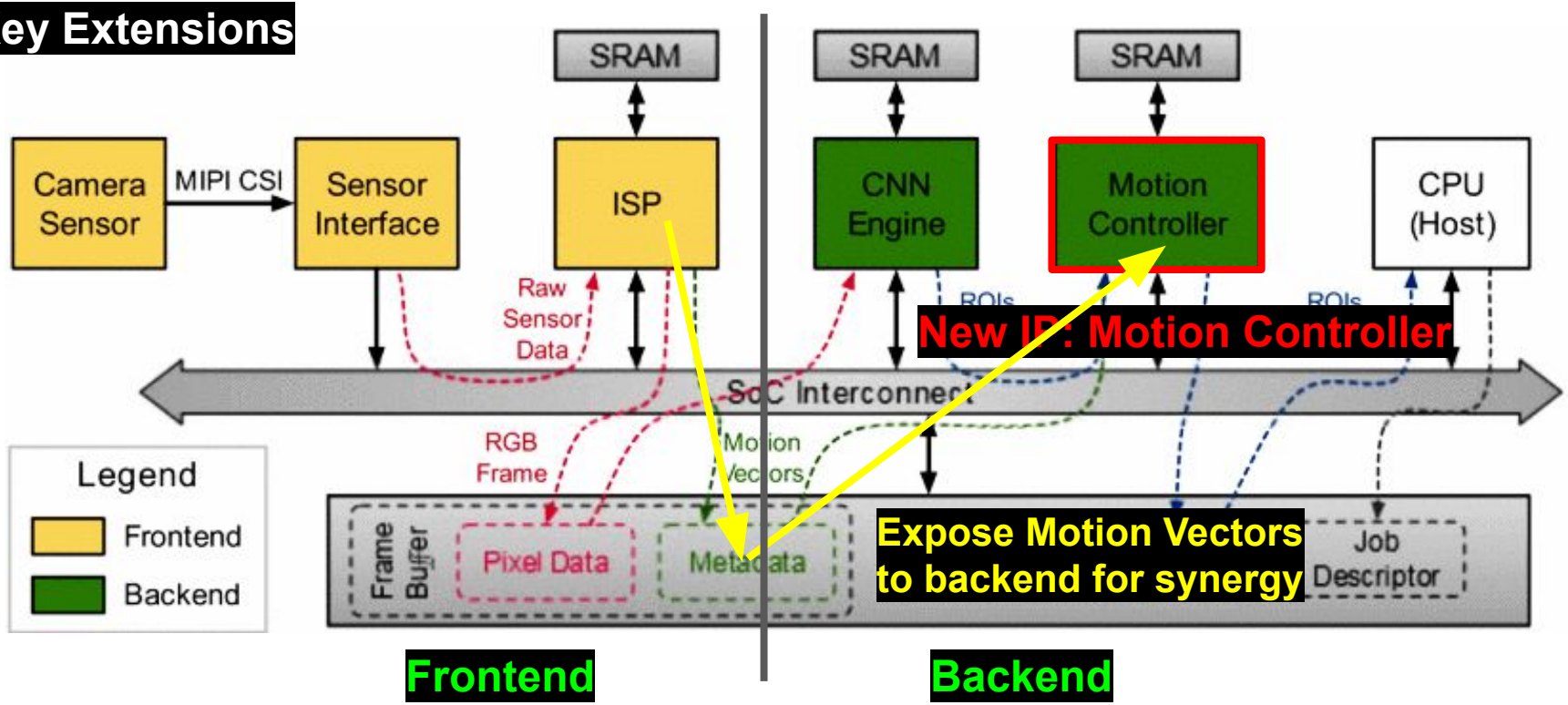
Principle 2. Extrapolation should be **decoupled from CNN** inference in the architecture.

Problem: CNN accelerators are still evolving rapidly. Tightly coupling with any particular CNN accelerator makes design inflexible.

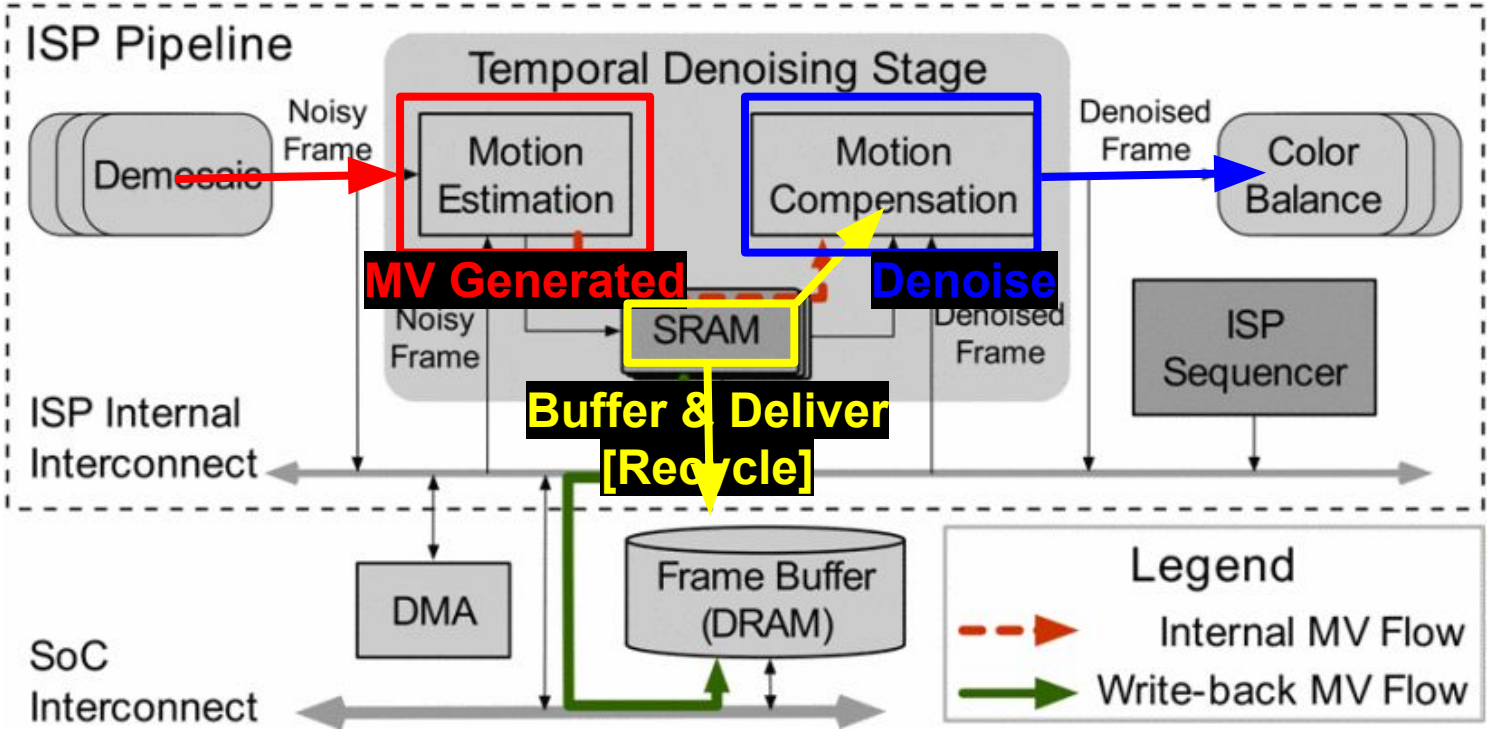
Solution: Propose **a new IP** instead of augmenting an existing CNN accelerator.

Architecture: Co-Designed Mobile SoC for CV

Key Extensions



Frontend: Temporal Denoising Stage



Frontend: Motion Vector Exposure Strategy

Two things to consider in ISP augmentation to expose MV:

1. By **what means** the MVs are exposed to system with **minimal design cost**

   → Piggyback frame buffer, deliver through DRAM as Metadata

2. How to **minimize the performance impact** on the ISP pipeline.

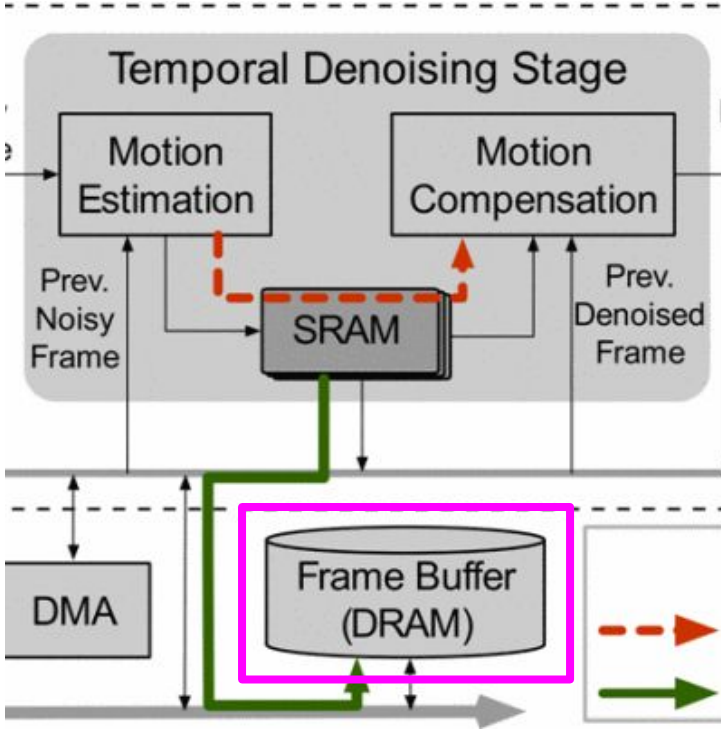
   → Take MV Traffic off the critical path

Frontend: Piggybacking Frame Buffer

Proposition: Expose MVs by storing in the metadata section of the frame buffer of DRAM.

Piggybacking the existing frame buffer mechanism gives **minimum design cost** with negligible memory traffic overhead.

Motion vector can be encoded in one byte. Very **small** compared to the pixel data already committed to the frame buffer.



Frontend: Taking MV Traffic off the Critical Path

Problem:

Reusing local SRAM as the DMA buffer for MV write-back

→ SRAM is precisely sized, SRAM resource contention

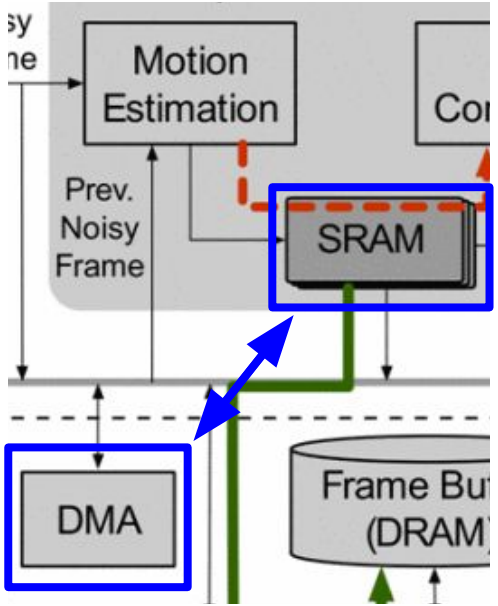
→ Stall the ISP pipeline.

Solution:

Double-buffer the SRAM in the TD stage

→ DMA engine opportunistically initiates the MV write-back

→ Effectively overlapping with the rest of the ISP pipeline.

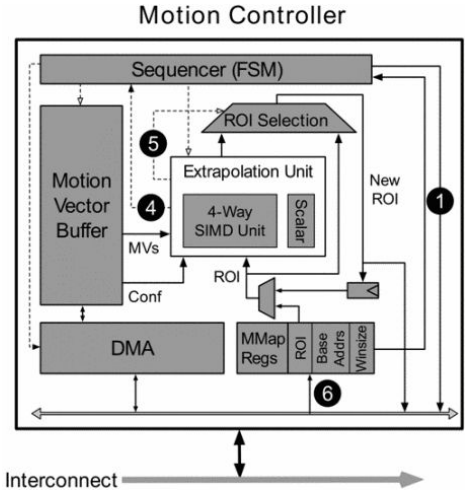


Backend: Motion Controller

New IP in the backend: **Motion controller**

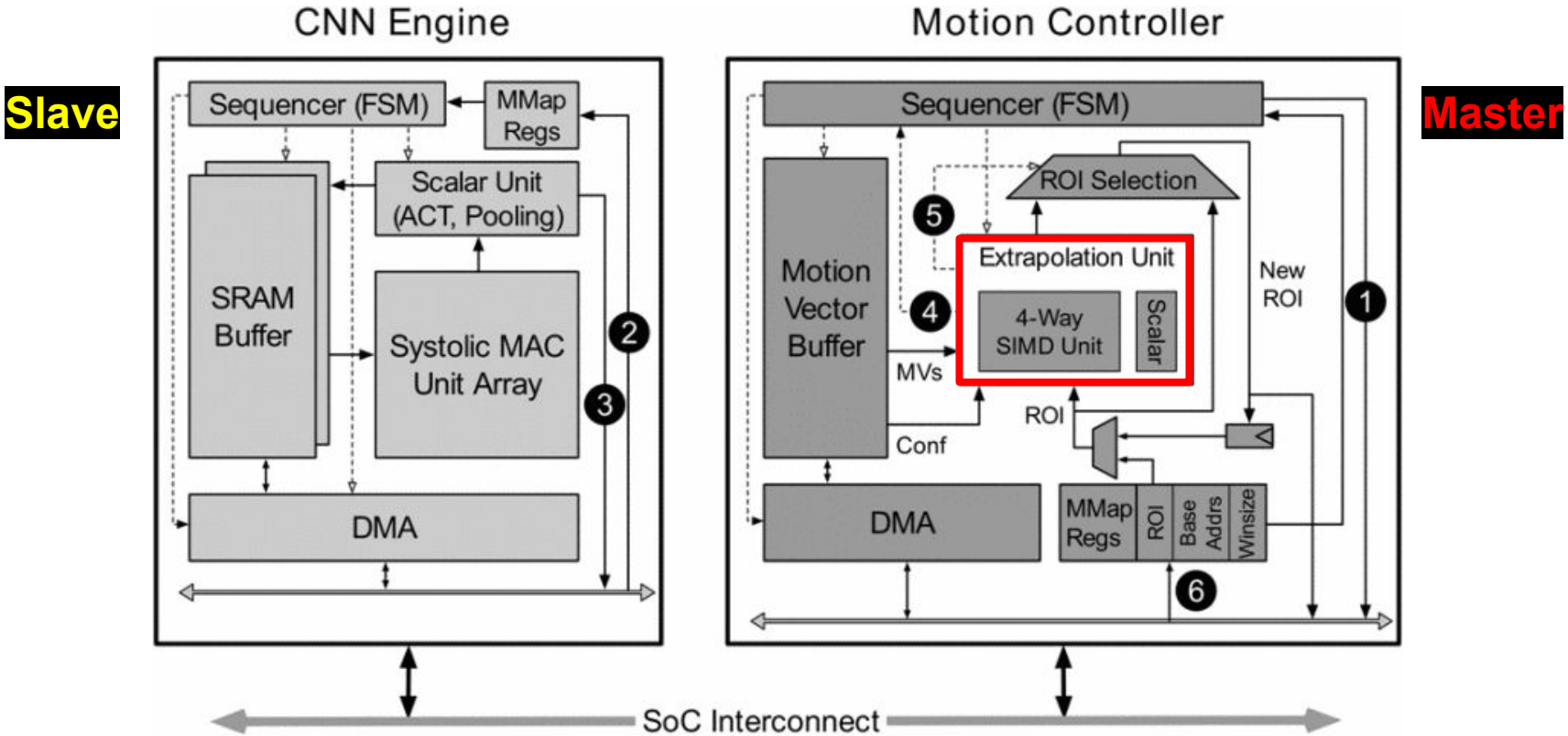
1. Executes the motion extrapolation algorithm.
2. Coordinates CNN engine without interrupting CPU

- Microcontroller (μC) based IP
  ~ Similar to sensor co-processors
- Equipped with an **on-chip SRAM** for Motion Vectors
- Programmable Sequencer Mechanism: **Reduces energy and area** compared to conventional mechanisms, while still providing programmability to control the datapath.



Motion Controller

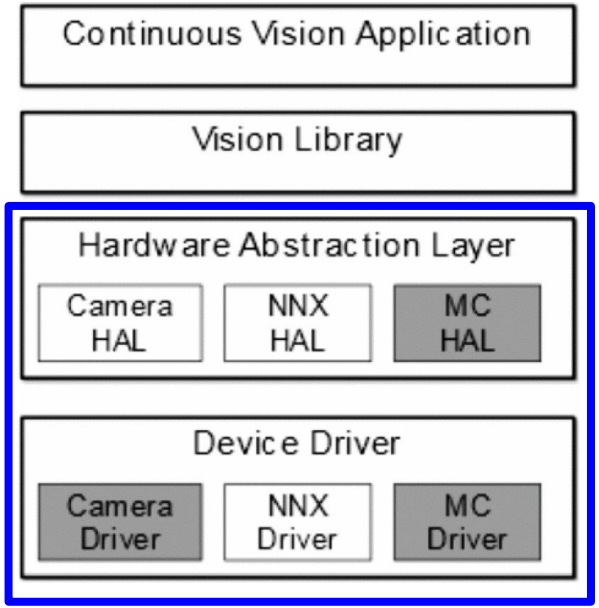
Backend: Motion Controller Microarchitecture



Backend: Software Implications

- No changes to programming interface and the CV libraries.
  → Enables software backward compatibility.

- Camera driver is enhanced to configure the base address of motion vectors
- MC HAL and MC Driver is added to support Motion Controller
- The rest is unmodified as MC takes Master role



Vision software stack

5. Implementation & Experimental Setup

Implementation: Hardware

- **Image Sensor:** AR1335
  - 1920 × 1080 (1080p) videos at 60 FPS
  - 180mW estimated power consumption

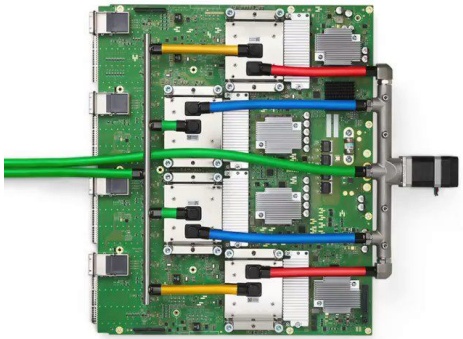
AR1335

- **ISP:** Jetson TX2 ISP
- 153mW estimated power consumption
- Simulator captures memory traffic information as Euphrates' ISP modification does impact memory traffic.

Implementation: Hardware

- **Neural Network Accelerator**
  - Customized systolic array-based CNN accelerator was integrated in the setup
    ~ Small version of Google TPU
  - 24×24 fully-pipelined Multiply-Accumulate array

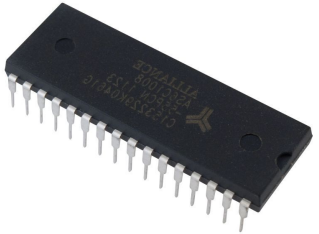
  - 1GHz Clock, 1.152 TOPS, 1.5MB SRAM, 1.58 mm^2
  - Power consumption: 651mW
  - Power-efficiency: 1.77 TOPS/W
  - Implemented in RTL



Google TPU

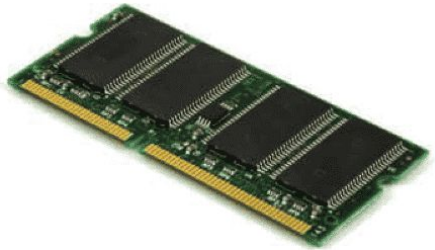
Implementation: Hardware

- **Motion Controller**
  - 4-wide SIMD datapath, 8KB local SRAM
    (can hold MV for single 1080p frame with 16×16MB size)
  - 100MHz Clock, 10ROIs per frame @ 60 FPS
  - Power consumption: 2.2 mW
  - Implemented in RTL



Static RAM

- **DRAM**
  - 8GB memory with 128-bit interface
  - Power consumption: 230mW @ 1080p, 60 FPS



Dynamic RAM

Experimental Setup: Software

Scenario 1: Object Detection Scenario

- **CNN1:** YOLOv2: Best accuracy & performance at the time: 3.4 TOPS
- **CNN2:** TinyYOLO: 20%⇩ accuracy & reduced requirement: 675 GOPS
- **Dataset:** In-house videos with extracted, annotated image sequences
  ~Similar to the Pascal VOC 2007 dataset

- **Accuracy Metric: IoU**(Intersection-over-Union) score
  - IoU: Intersection area / Union area of Predicted ROI & Ground Truth
- A detection is: **TP** if the IoU value is above threshold, otherwise **FP**
- Detection accuracy: **AP = TP/(TP+FP)** across all detections

Experimental Setup: Software

- **Scenario 2:** Visual Tracking Scenario
- **CNN3:** MDNet
- **Tracking scenarios:** Less compute-intensive, but often lack active cooling.
  → Increasing need to reduce the power/energy consumption

- **Dataset:** About 70,000 frames in total from two popular benchmarks
  - OTB 100: 100 videos with different visual attributes
  - VOT 2014: 25 image sequences with irregular bounding boxes
- **Accuracy metric:** Standard **success rate**
  - Success rate: % of detections with IoU ratio above threshold.

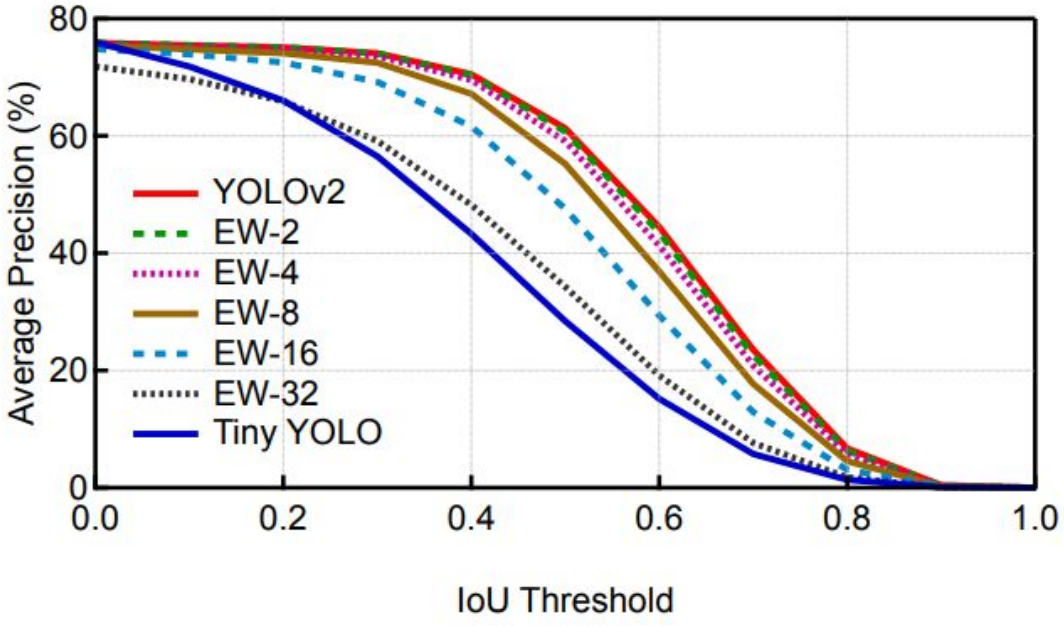
6. Evaluation

Measuring Effectiveness and Robustness

- Quantifiable Applications For Euphrates
  - Object Detection and Tracking
  - Critical for Continuous Vision Applications (Especially Mobile)
  - Prioritized by Commercial Intellectual Property Vendors
- Motion Estimation Sensitivity
  - Implementation Basis
  - Granularity Sensitivity
  - Quality Sensitivity

Object Detection Results - Measuring Accuracy

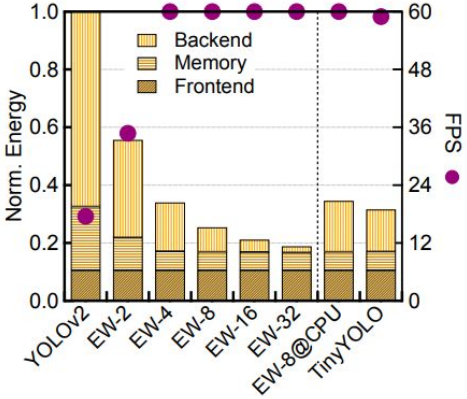
- Comparison
  - Average Precision
  - Euphrates vs. YOLOv2
- Variables
  - Window Size (EW-N)
  - Intersection-Of-Union
- Results
  - Accuracy Loss



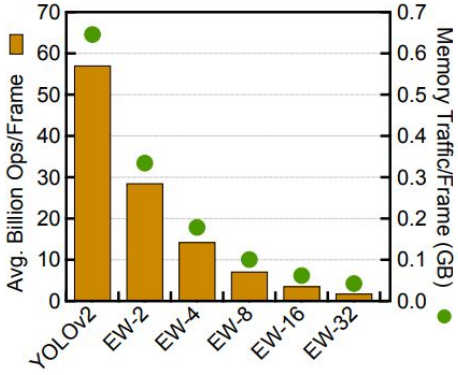
(a) Average precision comparison.

Object Detection Results - Energy / Performance

- Frontend
  - Constant FPS Configuration
  - Uniform Energy
- Main Memory & Backend
  - Baseline is Slow & Expensive
- Euphrates' Improvements
  - Relaxed Computation
  - Reduced SoC Memory Traffic
- Other Alternatives
  - Reducing CNN Complexity



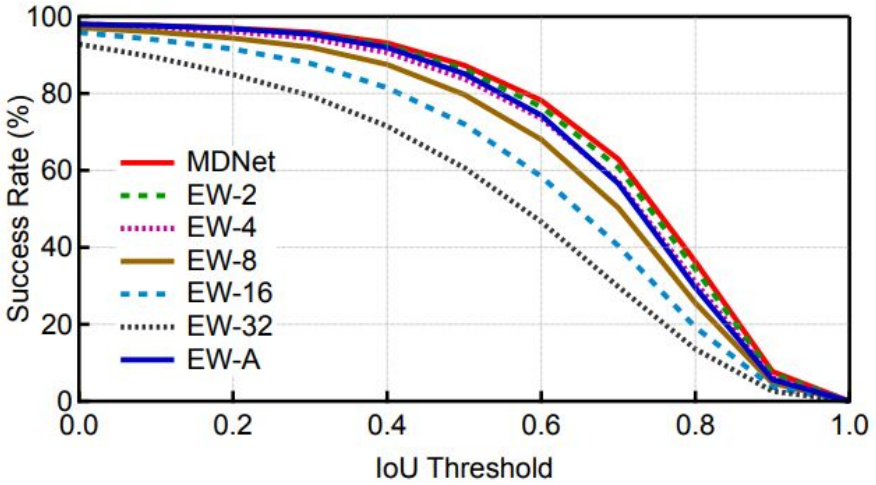
(b) Energy and FPS comparison.



(c) Compute and memory comparison.

Visual Tracking Results

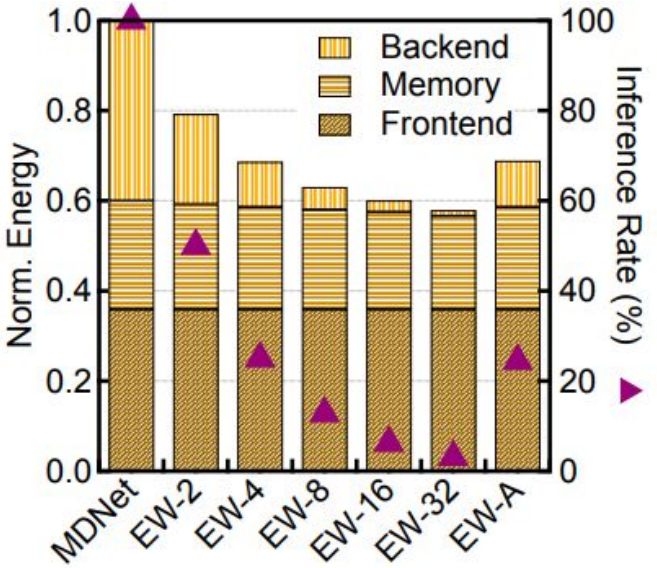
- Comparison
  - Success Rate
  - Euphrates vs. MDNet
- Variables
  - Window Sizes (EW-N / EW-A)
  - Intersection-Of-Union
- Results
  - Continued Trend



(a) Success rate comparison.

Visual Tracking Results - Energy

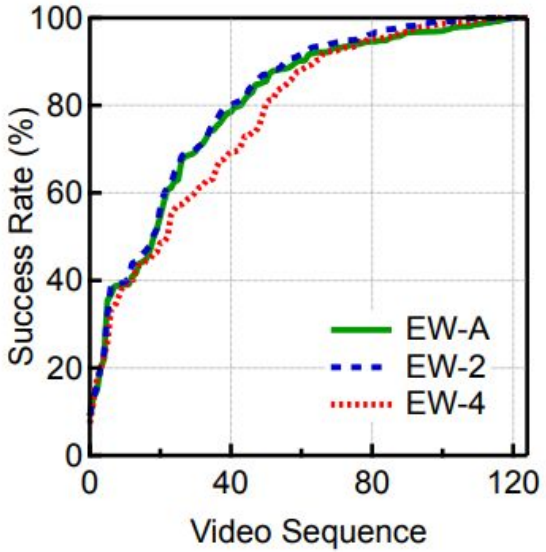
- Comparison
  - Less Significant Improvement
  - Energy vs. Inferences
- Trade-Off
  - Energy-Accuracy



(b) Normalized energy consumption and inference rate comparison.

Visual Tracking Results - Adaptive Mode

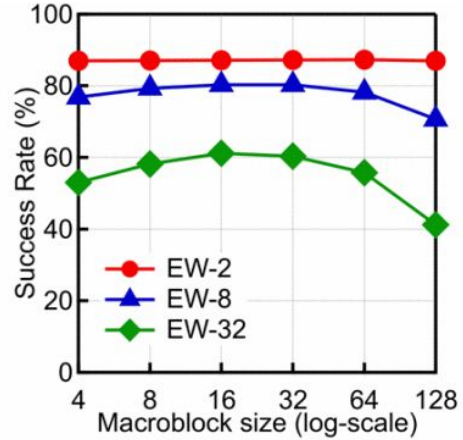
- EW-A vs. MDNet
  - Energy Improvements
  - Accuracy Loss
- EW-A vs. EW-4
  - Uniform Success Rate
  - Applicability



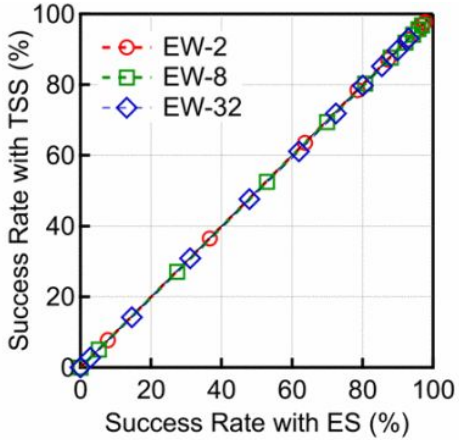
(c) Success rate for all 125 video sequences under an IoU ratio of 0.5.

Motion Estimation Sensitivity - Robustness

- Basis
- Granularity Sensitivity
  - Large MB
  - Small MB
  - Prefered MB
- Quality Sensitivity
  - SAD
  - Common Trade-Off
  - ES and TSS



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

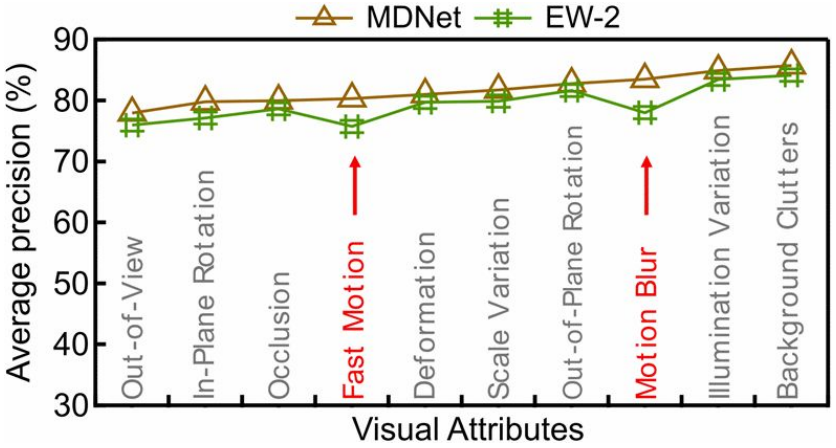


(b) Success rate comparison between ES and TSS.

7. Discussion

Accuracy sensitivity to different visual attributes.

- Motion Estimation Improvements
  - Fast Motion
  - Blurred Objects
  - Short Term
  - Long Term
- Hardware Design Alternatives
  - Video Codec



8. Related Work

Related Work

Motion-based Continuous Vision

- Prior works – CNN models for action recognition using motion vectors as inputs.
- Train the CNN directly using multiple frames.
- Euphrates – no extra training effort; achieves 0.2% higher accuracy.
- Fast YOLO requires training a separate CNN for motion prediction;
  - Euphrates leverages motion vectors.
- Fast YOLO – no extrapolation due to the lack of motion estimation;
  - Euphrates extrapolates using motion vectors.

→ SOTA CNNs can be used as baseline inference engine along with Euphrates.

Related Work

Energy-efficient Deep Learning

- Design goal of CNN architectures – reduce energy consumption
- Euphrates saves energy by reducing number of inferences

→ Motion extrapolation bridges the energy gap with little accuracy loss.

Specialized Imaging & Vision Architectures

- Advanced tasks (HDR and motion estimation) – performed by CPUs and GPUs.
- Modern processors perform advanced tasks in-situ – Eg: Google Pixel 2
- Future processors are expected to be highly programmable

→ More opportunities to co-design image processing and continuous vision systems.

Related Work

Computer Vision on raw sensor data

- CNN models can be trained using raw image sensor data
- RedEye and ASP Vision move early CNN layer(s) into the camera sensor

→ Motion can be estimated from raw sensor data using block matching

- Euphrates makes no assumption about image format of motion vectors
- Future work – port Euphrates to support raw data.

9. Conclusion

Key Takeaways

- Energy-efficient real-time computer vision is crucial for mobile systems
- Co-design mobile SoCs to overcome energy-efficiency barrier
- Leverage temporal motion information produced by ISP for motion estimation
- Reduce compute demand of CNN vision engine

Future scope:

- Explore cross-IP information sharing
- Expand co-design scope to on/off-chip components

Thank you!

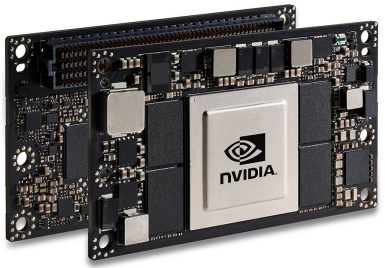
Q & A

- Any questions?

Implementation: Hardware Models

Simulator with Models to evaluate continuous vision pipeline.

- **Functional model:** Takes in video streams and implements the extrapolation algorithm.
- **Performance model:** Captures the timing behaviors of various vision pipeline components
- **Power model:** Provide energy estimation given information of cross-IP activities & SoC events



NVIDIA Jetson TX2

- Calibrated the models by measuring the Nvidia Jetson TX2 module