Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality

Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, Amit K. Roy-Chowdhury

ACM SenSys, November 10-13, 2019, Columbia University, New York, USA
Augmented Reality (AR)

• AR devices are forecast to be a $100 billion market by 2021

• AR is a killer app, with diverse applications
  • Pokemon Go, Google Translate, etc.

• In AR, virtual objects are overlaid onto real world objects to provide information
Motivation

- Object detection is an important task in the AR pipeline
  - Estimate object locations and their classes to overlay virtual holograms
- Deep Neural Networks (DNNs) yield highly-precise object detection but are energy-heavy on mobile devices
Design Goals

• **Precise classification** of real-world objects (e.g., cat vs dog)

• **Real-time, accurate tracking** of multiple, potentially moving objects → seamless user experience

• **Effective with other energy-saving techniques** such as mobile CPU throttling, mobile GPU, compressed DNNs

• Cope with **dynamic camera sensor inputs** due to handheld/wearable AR devices
Problem Statement

How can AR apps achieve good object detection and tracking performance and yet consume low energy?

• Key Idea
  • Interleave heavyweight DNNs with lightweight methods (object tracking and change detectors)

• Challenges
  • How to design trackers and change detectors that are lightweight yet effective?
  • How often to trigger lightweight methods?
  • How to cope with automatic CPU throttling?
Contributions

• Develop **MARLIN** framework to mediate between DNNs and lightweight methods

• Design a lightweight change detector to determine when to trigger DNNs

• Evaluate on Android smartphones with standard datasets and live experiments
  - *Dataset*: up to **73.3% energy savings**, losing at most 7.36% accuracy for most cases
  - *Live*: up to **81% energy savings** with negligible accuracy loss

• Compatible with a developer’s chosen DNN
  - E.g., Tiny YOLO, MobileNets, MobileNets w/ GPU, quantized (compressed) MobileNets
  - Up to **45.1% energy savings**, *beyond* what GPU or quantization already saves
Design of MARLIN Framework

- MARLIN Architecture
- MARLIN Manager
- Real-time Object Tracker
- Lightweight Change Detector
- DNN Object Detector
MARLIN is thrifty in triggering DNNs and only does so on a need-to-basis.
Real-time Object Tracker

• What tracking algorithm to use?
  • ORB image features + Lucas-Kanade optical flow
  • Real-time (update in < 10ms) and low power (0.2-0.3 W)

• How to know if the tracker failed?
  • Use normalized cross correlation (NCC) to estimate tracking accuracy
    • Because we do not know the ground truth object locations \textit{a priori}
    • NCC measures object feature similarity between two frames
Lightweight Change Detector

• **Requirements**: real-time, low-power, low false positive rates, ignore existing objects

• **Existing methods**: Background subtraction is susceptible to camera motion \( \rightarrow \) frequent DNN triggers \( \rightarrow \) high energy consumption

• **Our method**: Random forest with image features (color histogram)
  • Fast (~ 4ms), low-power ( < 0.1 W), accurate compared to other lightweight ML techniques
  • Color existing objects with a “white box” to avoid triggering changes on them

<table>
<thead>
<tr>
<th>ML techniques</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forest with color histogram (MARLIN)</td>
<td>88.0</td>
<td>81.7</td>
</tr>
<tr>
<td>SVM using HOG features</td>
<td>64.9</td>
<td>61.4</td>
</tr>
</tbody>
</table>

To avoid triggering on existing objects, they are “whited out”.
DNN Object Detector

• DNNs provide high classification and detection accuracy
  • extract image features automatically
  • pass through convolutional layers
  • output class labels + object locations

• We use off-the-shelf DNNs
• They can be plugged into MARLIN to save energy

<table>
<thead>
<tr>
<th>Trained and tested DNNs</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLO</td>
<td>-</td>
</tr>
<tr>
<td>Tiny YOLO</td>
<td>TYL</td>
</tr>
<tr>
<td>MobileNets</td>
<td>MNet</td>
</tr>
<tr>
<td>GPU-assisted MobileNets</td>
<td>MNet-GPU</td>
</tr>
<tr>
<td>Quantized MobileNets</td>
<td>MNet-Q</td>
</tr>
</tbody>
</table>
MARLIN Evaluations

• Offline evaluations
  • Comparing multiple DNN baseline approaches
  • Comparing with the best baseline
  • Case study: zoomed-in video
  • Impact of mobile CPU throttling

• Online (live) evaluations
**Experimental Setup**

- **Devices**: Google Pixel 2 running Android 8.0 and LG G6 running Android 7.0 (for automatic CPU throttling)

- **Datasets**
  - ImageNet-Video for offline experiments
  - VOC-2007, VOC-2012, and Penn-Fudan Pedestrian to train DNN for live experiments

- **Training/validation/test**
  - 350 videos for offline training and validation
  - 15-80 other videos for online testing

- **Baseline**
  - Continuous executions of DNN (DNN triggered immediately after previous one)
How much energy does MARLIN save?

MARLIN saves power consumption by **45.1%**, compared to baseline quantized DNN or GPU-assisted DNN, while suffering **8.3%** accuracy loss.

- In terms of accuracy, Tiny YOLO (TYL) is the best baseline to compare with MARLIN.
Comparing with the best baseline (Tiny YOLO)

- **MARLIN** extends battery life by **1.85x**, with a small accuracy loss.

- In terms of relative accuracy per video, \( \frac{\text{Accuracy}_{\text{TYL}} - \text{Accuracy}_{\text{MARLIN}}}{\text{Accuracy}_{\text{TYL}}} \) for **46.3%** of the videos, MARLIN can even improve accuracy!
Surprisingly, accuracy increases sometimes... why?
A case study of a zoom-in video

- Baseline Tiny-YOLO sometimes yields false results due to frequently executing DNNs and tracking bad features
- MARLIN tracks good, stable image features found by the object tracker and change detector
- For this video, MARLIN has 55% ACP accuracy gain, and saves $2.5 \text{ W}$ of power and extends battery life by 3.5 hours!

Classification Precision = 0.5

Classification Precision = 1.0

* ACP = Average Classification Precision
Automatic CPU throttling can save energy – Does MARLIN still help?

- Default throttled CPU → tracking latency increased → tracking accuracy reduced
- MARLIN reduces DNN executions → fewer throttling instances → improved tracking accuracy and reduce power consumption

* ACP = Average Classification Precision
  IOU = Intersection Over Union
Live experiments

<table>
<thead>
<tr>
<th>Sets</th>
<th>Methods</th>
<th>IOU (%)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live 1</td>
<td>Tiny-YOLO</td>
<td>61</td>
<td>1.724</td>
</tr>
<tr>
<td></td>
<td>MARLIN</td>
<td>61</td>
<td>0.319</td>
</tr>
<tr>
<td>Live 2</td>
<td>Tiny-YOLO</td>
<td>56</td>
<td>1.710</td>
</tr>
<tr>
<td></td>
<td>MARLIN</td>
<td>51</td>
<td>0.880</td>
</tr>
</tbody>
</table>

- AR user holds two Google Pixel 2 phones
  - One device runs Tiny YOLO, the other MARLIN
  - Track 2-3 people in the field-of-view

- MARLIN achieves similar tracking accuracy to Tiny-YOLO, while using only ~20% (Live 1) and ~50% (Live 2)* of the power

*In Live 2, there are more subjects with more movements than Live 1.
Conclusions

• **Energy consumption** is a major concern for AR
  • Battery percentage drops 45% after 60 minutes

• We design MARLIN as power-thrifty framework for object detection and tracking for AR that is **compatible with multiple DNNs**

• MARLIN intelligently alternates between DNNs and lightweight methods to achieve high accuracy while **saving ~45% energy**

• **Future work** includes using inertial odometry to further save energy
Thank you.
Backup Slides
These cases allow MARLIN to maintain good accuracy by triggering DNN when needed.

- **track_status < THRES**
  - DNN is ready: Start DNN thread
  - DNN is not ready: Skip DNN, this case allows MARLIN to save energy significantly

- **track_status >= THRES**
  - change_status == True: Start DNN thread
  - change_status == False: Skip DNN, this case allows MARLIN to save energy significantly
**MARLIN Architecture**

- **MARLIN Manager (MM)** receives frame from camera and feed it to object tracker and/or change detector, which provide feedback to MM to decide whether or not to feed the frame to DNN (energy-heavy but may recover the system from low accuracy).

  - MM first looks at `track_status` (how much a tracked object change in appearance compared to that in a previous frame)
    - If `track_status` < threshold, check DR flag and send this frame to DNN
    - Otherwise, send this frame to change detector and if there is a significant change (e.g. likely to have objects of interest in the scene) outside of the tracked object, check DR and send this frame to DNN
Real-time Object Tracker

• ORB feature extraction can be done in near-real-time
• Object tracking by optical flow (of ORB features)
  • Extraction + tracking can be done < 10ms with 0.2-0.3 W, so this is practical for a 30-fps 640x480 camera
• Calculate normalized cross correlation (NCC) of a tracked object from a previous to the current frame
  • NCC is a good estimation of tracking quality (NCC is low when there is an occlusion or object deformation)
  • Ex. NCC from frame 1 to 2 = 0.92, and NCC from 2 to 3 = 0.69
  • Send this information as track_status to MM

Frame 1  Frame 2  Frame 3
Lightweight Change Detector

• A supervised learning agent that takes a vector of floating-point numbers (compressed from an image frame) and returns a binary decision as change_status back to MM

• Input vector represents color features in the frame after the areas of the tracked objects have been removed (whited out)

• Train the agent to learn color features of foreground (tiger or elephant etc.) and background (sky or grass etc.)

• It uses random forest (the best among different ML techniques tested), consisting of 50 decision trees

• It works very fast (~ 4ms per frame) and is very low-powered ( < 0.1 W)