DeepCache: Principled Cache for Mobile Deep Vision

Mengwei Xu\textsuperscript{1}, Mengze Zhu\textsuperscript{1}, Yunxin Liu\textsuperscript{2}
Felix Xiaozhu Lin\textsuperscript{3}, Xuanzhe Liu\textsuperscript{1}

\textsuperscript{1}Peking University, \textsuperscript{2}Microsoft Research, \textsuperscript{3}Purdue University
Background: Mobile Vision

- Your mobile device sees what you see, and does what you cannot do
  - Core: computer vision algorithm

Augmented Reality

Game

Recognition & Detection

Face Beauty
Background: CNN-based Vision

• Convolutional Neural Network (CNN) is the state-of-the-art vision algorithm.

  CNN model: a graph of computation nodes (convolution, pooling, activation, etc)

  CNN model: a graph of computation nodes (convolution, pooling, activation, etc)

  convolution operation (input feature map * kernel = out feature map)

• CNN is accurate, but also resource-hungry.
Background: Optimizing CNN Workloads

- Our approach: leveraging the temporal locality of mobile video stream
- Similar but not identical
  - Object movement/appearance
  - Camera movement
  - Light variation
  - etc...

Algorithm-level Compression
- quantization
- pruning
- factorization
- distilling

Hardware-level Acceleration
- CPU
- GPU
- DSP
- AI-specific chips

previous frame

Current frame
Caching Mobile Vision – a naïve approach

• Just cache/reuse the final results based on input image
Caching Mobile Vision – a naïve approach

• Just cache/reuse the final results based on input image
Caching Mobile Vision – a naïve approach

• Just cache/reuse the final results based on input image
Caching Mobile Vision – a naïve approach

• Just cache/reuse the final results based on input image
Caching Mobile Vision – a naïve approach

• Just cache/reuse the final results based on input image

• Why it’s not enough?
  • Coarse-grained: whole image as comparison unit
  • Cannot handle position-sensitive tasks

Two images are similar
• Similar background
• Similar animals
But the elephant position is different!
Caching Mobile Vision – *DeepCache*

- Treat image as a collection of blocks, and cache/reuse them at a fine granularity.

**KEY IDEA**

Reuse the CNN computations of similar regions

- previous frame
- current frame
Caching Mobile Vision – **DeepCache**

- Treat image as a collection of blocks, and cache/reuse them at a fine granularity.
Challenges of DeepCache

• **Scene variation** – the overall background may be moved between frames
  - Moving camera, autonomous driving, drone, etc

![Previous frame](image1.png) ![Current frame](image2.png)
Challenges of DeepCache

- **Cache erosion** – reusability tends to diminish at its deeper layers

![Diagram showing cache erosion and re-computation required with 3x3 convolution](image-url)
Challenges of DeepCache

- **Cache erosion** – reusability tends to diminish at its deeper layers

1. Merge smaller ones

2. Good news: early layers contribute most of the computation cost and also suffer less cache erosion.

![Diagram showing challenges of DeepCache](image-url)
DeepCache Design: Overview

**Design Principles**
- No cloud offloading
- No efforts from developers
- No modification to models

**Two modules**
- Image matcher
- Cache-aware inference engine
DeepCache Design: Image Matching

• Principles: high similarity, low overhead, and merged to big ones.
• Input: two raw images
• Output: a set of matched rectangles
  • \((x_1, y_1, w, h)\) in current frame -> \((x_2, y_2, w, h)\) in previous frame
DeepCache Design: Image Matching

- Step 1: dividing the current frame into an \( N \times N \) grid.
  - \( N \) is a configurable parameter (default: 10 \( \times \) 10).
DeepCache Design: Image Matching

• Step 2: find the most-matched block in previous frame for each divided block
  • Motion estimation: diamond search
DeepCache Design: Image Matching

• Step 3: calculate the average block movement (offset): \((M_x, M_y)\).
  • Filter those outliers

\[
(M_x, M_y) = \left( \frac{\sum(x'_i - x_i)}{K}, \frac{\sum(y'_i - y_i)}{K} \right), \langle (x_i, y_i), (x'_i, y'_i) \rangle \in S
\]
DeepCache Design: Image Matching

• Step 4: calculate the similarity between block \((x_1, y_1)\) in current frame and the block \((x_1+M_x, y_2 + M_y)\) with average movement in previous frame
  • Metrics: Peak Signal to Noise Ratio (PSNR)

![Diagram showing the calculation of similarity between blocks in previous and current frames.](image-url)
DeepCache Design: Image Matching

• Step 5: merge blocks into larger ones if possible

$\text{Current \ Frame} = (x_1 + M_x, y_2 + M_y)$

$\text{Previous \ Frame} = (x_1, y_1)$
DeepCache Design: Image Matching

- Optimization 1: skip block matching in Step 2 (k-skip)

- Optimization 2: in Step 4, reuse the matching scores computed in Step 2
  - Not always applicable: depends on the average movement
DeepCache Design: Cache-aware CNN Inference

- **Propagation**: the reusable regions passed from image matching is not unchangeable during execution among CNN layers.

Because of cache erosion!
But what affects cache erosion?
DeepCache Design: Cache-aware CNN Inference

- **Propagation**: the reusable regions passed from image matching is not unchangeable during execution among CNN layers.

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Layer Parameters</th>
<th>Output($D_t$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>kernel=k x k</td>
<td>$x' = \lceil(x + p)/s\rceil$, $y' = \lceil(y + p)/s\rceil$</td>
</tr>
<tr>
<td></td>
<td>stride=s,</td>
<td>$w' = \lfloor(w - k)/s\rfloor$, $h' = \lfloor(h - k)/s\rfloor$</td>
</tr>
<tr>
<td></td>
<td>padding=p</td>
<td></td>
</tr>
<tr>
<td>Pooling</td>
<td>radius=r</td>
<td>$x' = x + r$, $y' = y + r$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$w' = w - 2 \times r$, $h' = h - 2 \times r$</td>
</tr>
<tr>
<td>LRN [10]</td>
<td>radius=r</td>
<td></td>
</tr>
<tr>
<td>Concat [7]</td>
<td>input number=N</td>
<td>overlapped region of these $N$ rectangles</td>
</tr>
<tr>
<td>Fully-connect</td>
<td></td>
<td>$(x', y', w', h') = (0, 0, 0, 0)$</td>
</tr>
<tr>
<td>Softmax [16]</td>
<td></td>
<td>$(x', y', w', h') = (x, y, w, h)$</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- partial erosion
- full erosion
- no erosion
DeepCache Design: Cache-aware CNN Inference

- **Why Propagation?** Why not match the input of each layer?
  - Low return: feature maps are high dimensional data, difficult to interpret.
  - High cost: matching feature map requires a lot of computations (40× compared to propagation for ResNet).

![Normalized Latency Graph]

- **DeepCache**: match input images once, and using propagation later
- **MIL**: matching inter-layer
DeepCache Design: Cache-aware CNN Inference

- **Cache/Reuse**: reuse the computation results at the output of convolutions.

  - **Mind the data locality during reuse!**
    - Depends on the convolution implementation: `img2col + gemm`, unrolled, etc.
DeepCache Implementation

• Image matching is implemented based on *RenderScript*
  • A programming framework on Android for intensive computations
  • GPU-support, generic, high data-parallel

• Cache-awareness is built upon *ncnn*
  • Popular deep learning inference framework for mobile devices
  • High speed, lightweight, no dependency
Evaluation – Setup

• Popular CNN models and datasets

<table>
<thead>
<tr>
<th>Application</th>
<th>Model Name</th>
<th>Architecture</th>
<th># of Conv</th>
<th>Model Output</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity recognition</td>
<td>REC_1</td>
<td>AlexNet [43]</td>
<td>5</td>
<td>human activity type</td>
<td>UCF101 [57]</td>
</tr>
<tr>
<td></td>
<td>REC_2</td>
<td>GoogLeNet [58]</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REC_3</td>
<td>ResNet-50 [35]</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object detection</td>
<td>DET</td>
<td>YOLO [54]</td>
<td>8</td>
<td>object types and positions</td>
<td></td>
</tr>
<tr>
<td>Self-driving</td>
<td>DRV</td>
<td>Dave-origin [5, 22]</td>
<td>5</td>
<td>steering angle</td>
<td>Nvidia driving dataset [12]</td>
</tr>
</tbody>
</table>

• Platform: Nexus 6, Android 6.0

• Alternative
  • ncnn without cache
  • Coarse-grained cache used in [1] DeepMon

Evaluation – Execution Speedup

- DeepCache saves 15% ~ 28% model execution time (2X DeepMon)
  - The speedup depends on model architecture
  - Deeper layers, less savings

<table>
<thead>
<tr>
<th>Application</th>
<th>Model Name</th>
<th>Architecture</th>
<th># of Conv</th>
<th>Model Output</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity recognition</td>
<td>REC_1</td>
<td>AlexNet [43]</td>
<td>5</td>
<td>human activity type</td>
<td>UCF101 [57]</td>
</tr>
<tr>
<td></td>
<td>REC_2</td>
<td>GoogLeNet [58]</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REC_3</td>
<td>ResNet-50 [35]</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object detection</td>
<td>DET</td>
<td>YOLO [54]</td>
<td>8</td>
<td>object types and positions</td>
<td></td>
</tr>
<tr>
<td>Self-driving</td>
<td>DRV</td>
<td>Dave-orig [5, 22]</td>
<td>5</td>
<td>steering angle</td>
<td>Nvidia driving dataset [12]</td>
</tr>
</tbody>
</table>
Evaluation – Energy Saving

- DeepCache can save around 20% energy consumption of processing same number of images.
  - The energy saved by using DeepCache to process 10 images (ResNet) is equal to 40 seconds of playing video on mobile devices.
Evaluation – Accuracy

- DeepCache is able to keep the accuracy
  - 2% on average, similar to DeepMon
Evaluation – Memory Overhead

• The overhead of cache is acceptable
  • 2MB ~ 44MB, while nowadays mobile devices usually have more than 1G memory
  • Note we only cache the results of convolutional layers, and only for one frame
Conclusion

• DeepCache: cache design for mobile deep vision
  • Image matching on raw images
  • Cache/reuse in inference engine

• Evaluation
  • ~20% execution speedup and energy savings
  • Little accuracy loss

Thank you for attention!
Evaluation – Execution Speedup

- DeepCache saves 15% ~ 28% model execution time (> DeepMon)
  - Performance depends on scenarios

T1: Basketball
T2: ApplyEyeMakeup
T3: CleanAndJerk
T4: Billiards
T5: BandMarching
T6: ApplyLipstick
T7: CliffDiving
T8: BrushingTeeth
T9: BlowDryHair
T10: BalanceBeam
Evaluation – Configuration

• Some configuration can be tailored to make better trade-off among accuracy, latency, and energy for different scenarios and applications.
  • Matching threshold, block size