# DeepCache: Principled Cache for Mobile Deep Vision

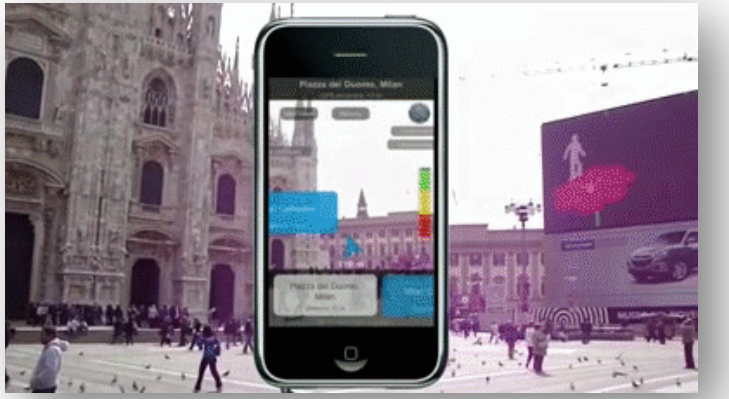
**Mengwei Xu**[1], Mengze Zhu[1], Yunxin Liu[2]
Felix Xiaozhu Lin[3], Xuanzhe Liu[1]

[1]Peking University, [2]Microsoft Research, [3]Purdue University



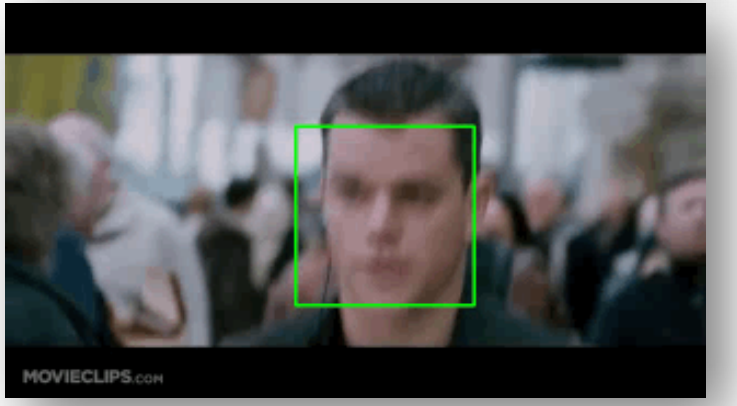
# Background: Mobile Vision

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



*Augmented Reality*

*Game*

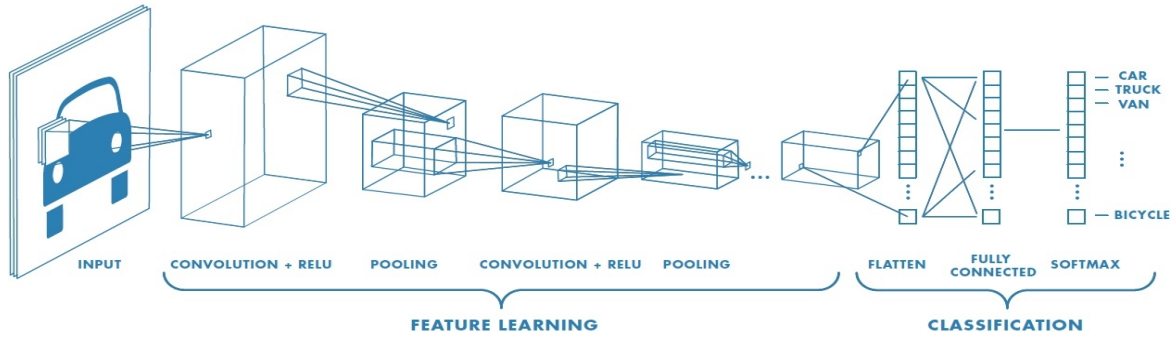



*Recognition & Detection*

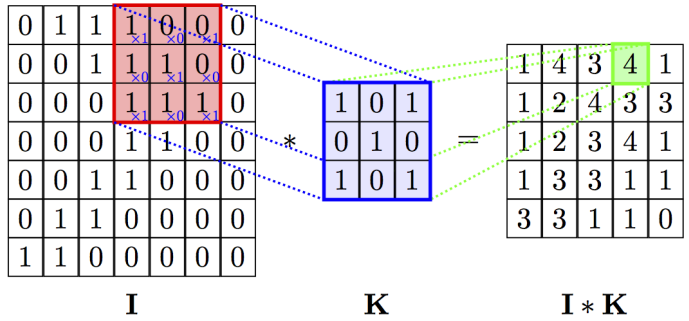

*Face Beauty*

# Background: CNN-based Vision

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



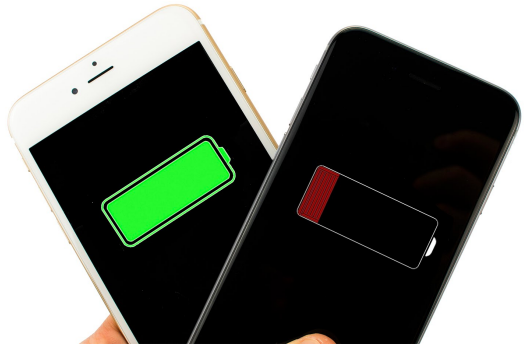

*CNN model: a graph of computation nodes (convolution, pooling, activation, etc)*




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

- CNN is accurate, but also resource-hungry.

| | #conv. layers | #MACCs [millions] | #params [millions] | #activations [millions] | ImageNet top-5 error |
|---|---|---|---|---|---|
| **ZynqNet CNN** | **18** | **530** | **2.5** | **8.8** | **15.4%** |
| AlexNet | 5 | 1 140 | 62.4 | 2.4 | 19.7% |
| Network-in-Network | 12 | 1 100 | 7.6 | 4.0 | ~19.0% |
| VGG-16 | 16 | 15 470 | 138.3 | 29.0 | 8.1% |
| GoogLeNet | 22 | 1 600 | 7.0 | 10.4 | 9.2% |
| ResNet-50 | 50 | 3 870 | 25.6 | 46.9 | 7.0% |
| Inception v3 | 48 | 5 710 | 23.8 | 32.6 | 5.6% |
| Inception-ResNet-v2 | 96 | 9 210 | 31.6 | 74.5 | 4.9% |
| SqueezeNet | 18 | 860 | 1.2 | 12.7 | 19.7% |
| SqueezeNet v1.1 | 18 | 390 | 1.2 | 7.8 | 19.7% |

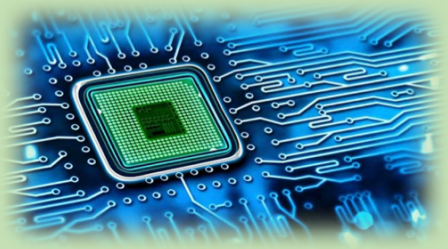
# Background: Optimizing CNN Workloads

## Algorithm-level Compression

- quantization
- pruning
- factorization
- distilling

## Hardware-level Acceleration

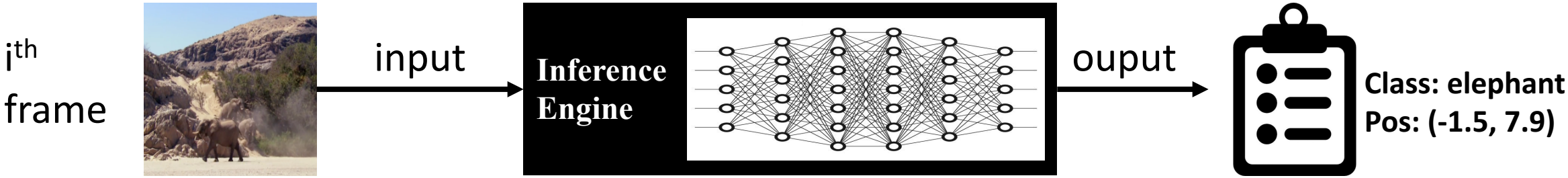
- CPU
- GPU
- DSP
- AI-specific chips



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

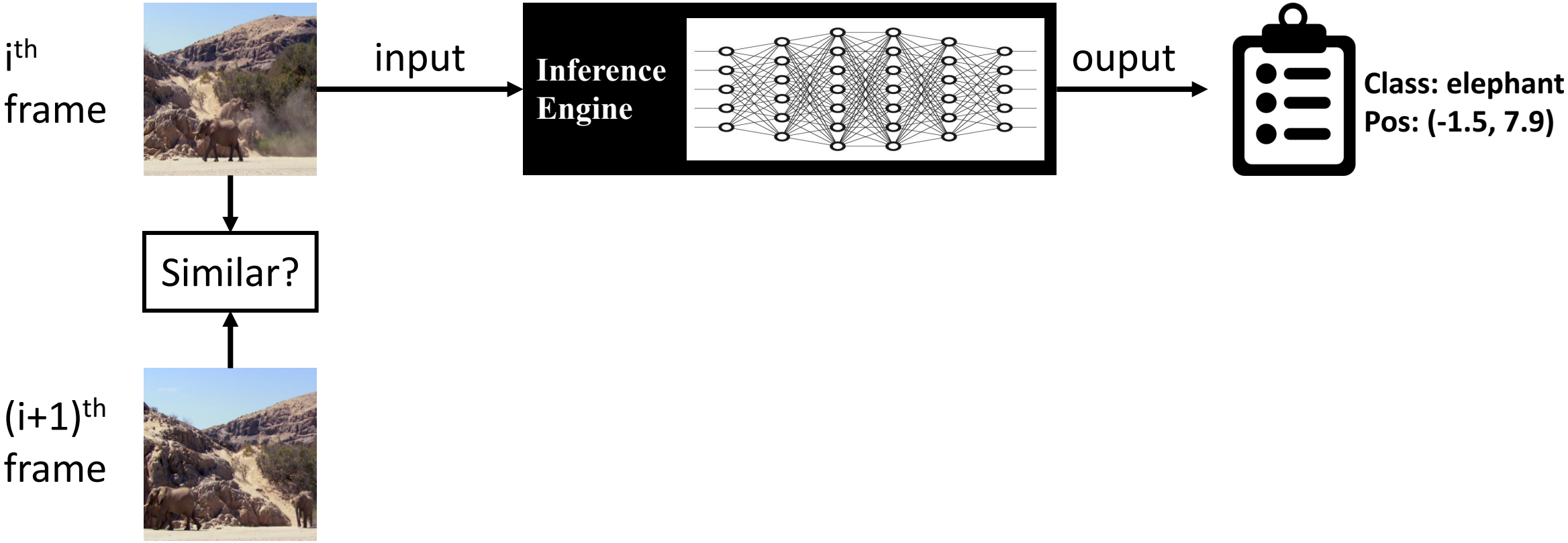

# Caching Mobile Vision – a naïve approach

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



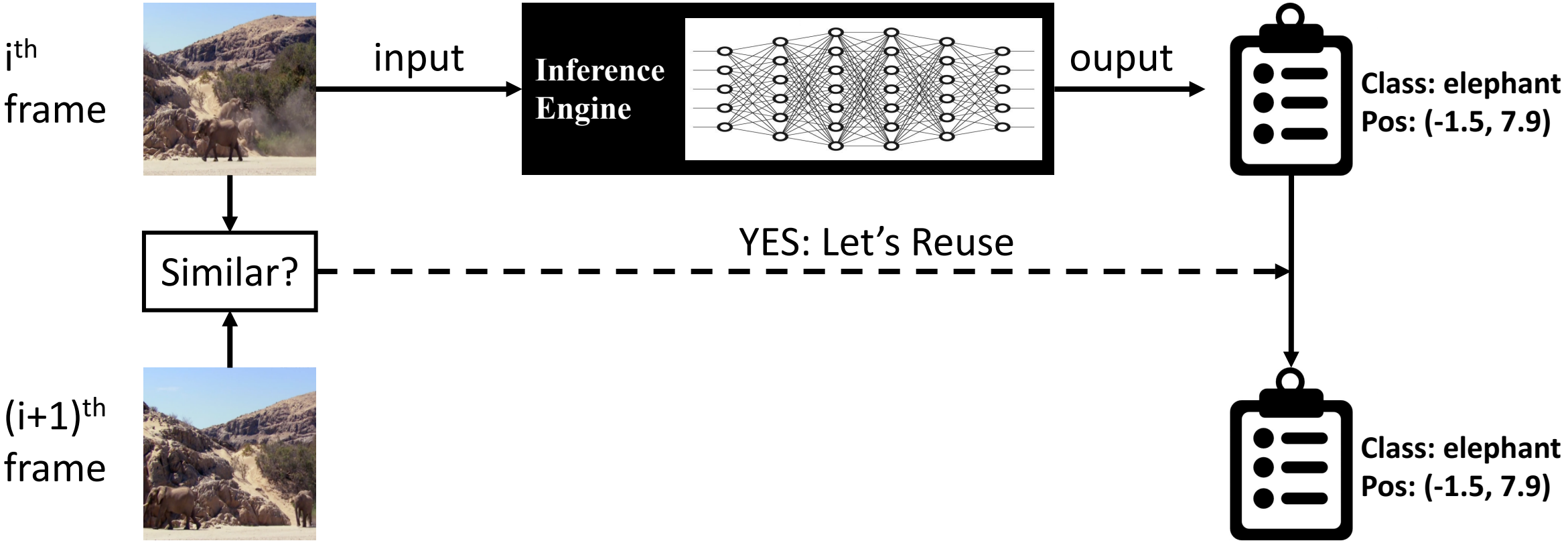
# Caching Mobile Vision – a naïve approach

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



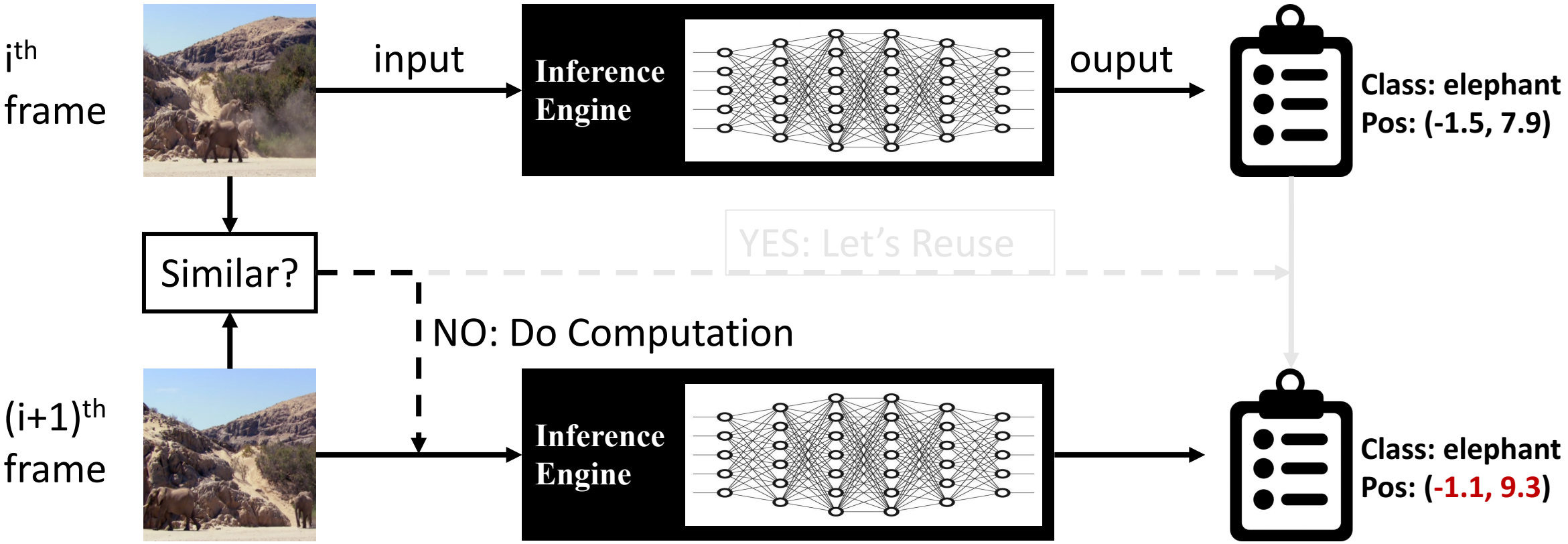
# Caching Mobile Vision – a naïve approach

- 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*

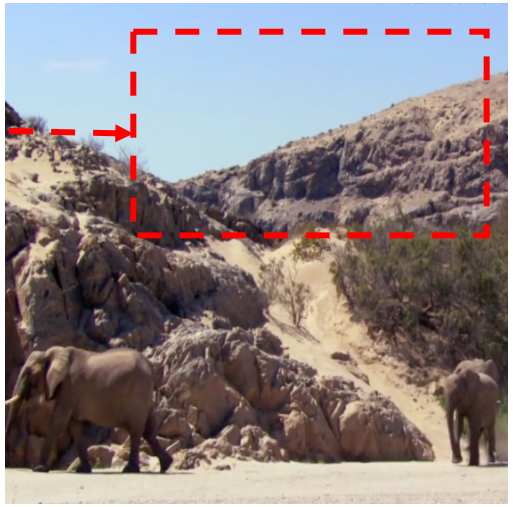
- Treat image as a collection of blocks, and cache/reuse them at a fine granularity.

**KEY IDEA**

previous frame

cache & reuse

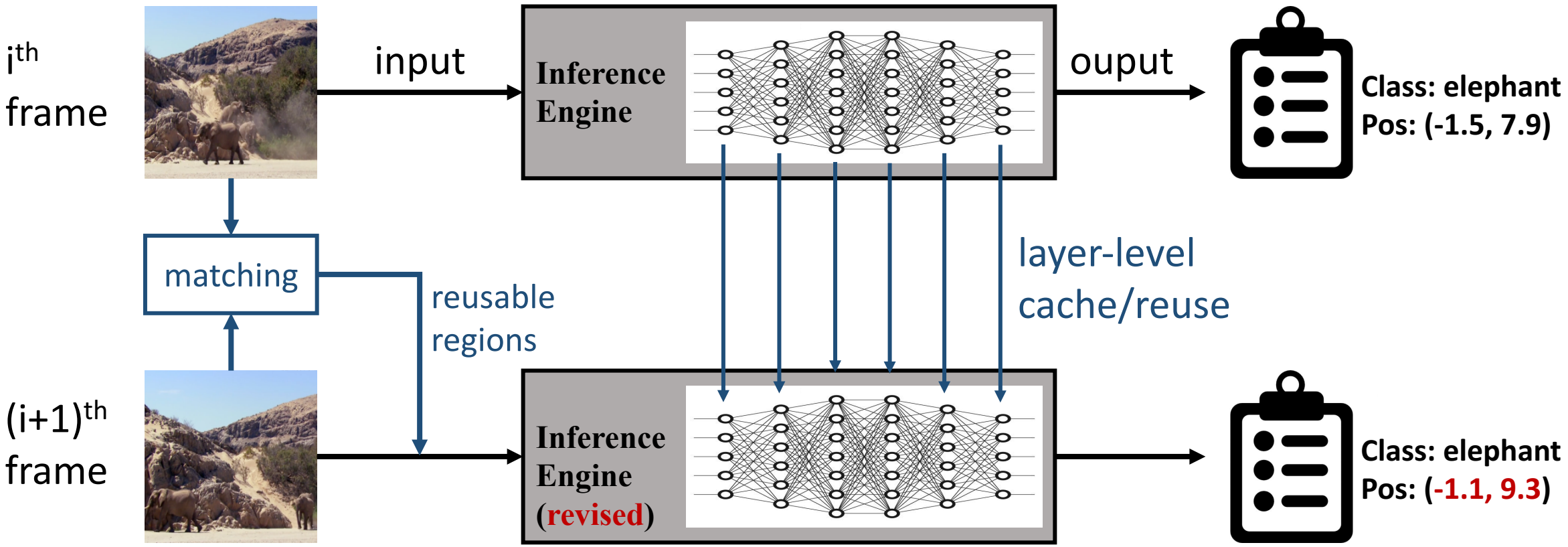
Reuse the CNN computations of similar regions



current frame

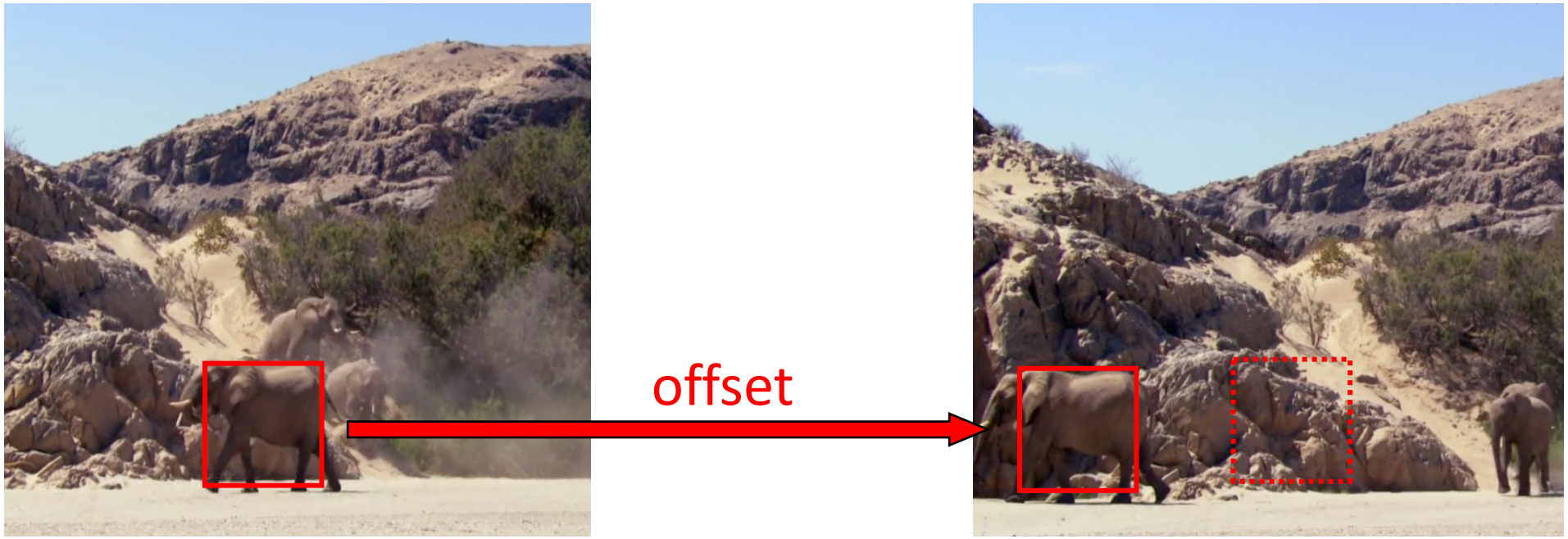
# Caching Mobile Vision – *DeepCache*

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



# Challenges of DeepCache

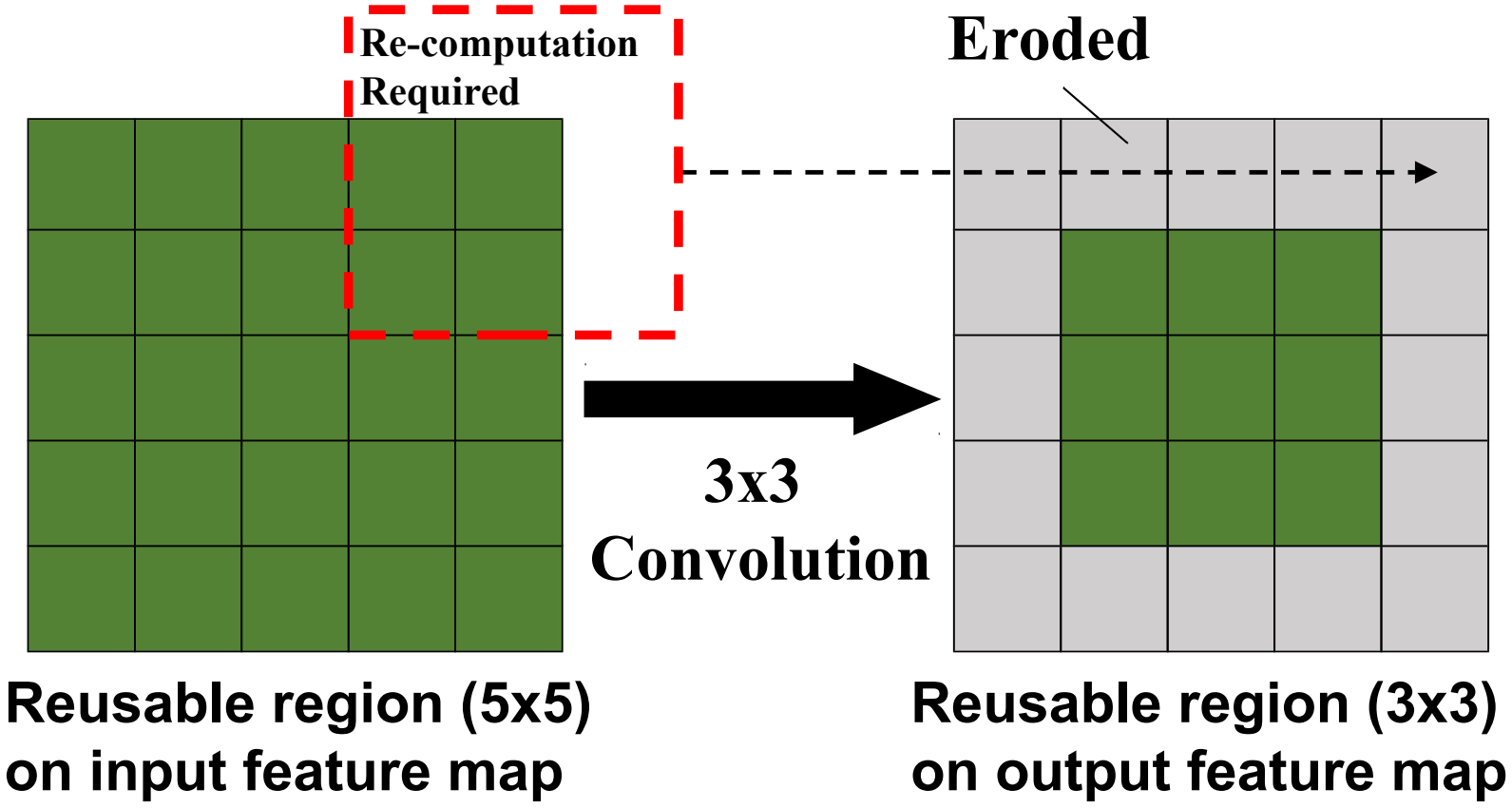
- **Scene variation** – the overall background may be moved between frames
  - Moving camera, autonomous driving, drone, etc




**previous frame** **current frame**

# Challenges of DeepCache

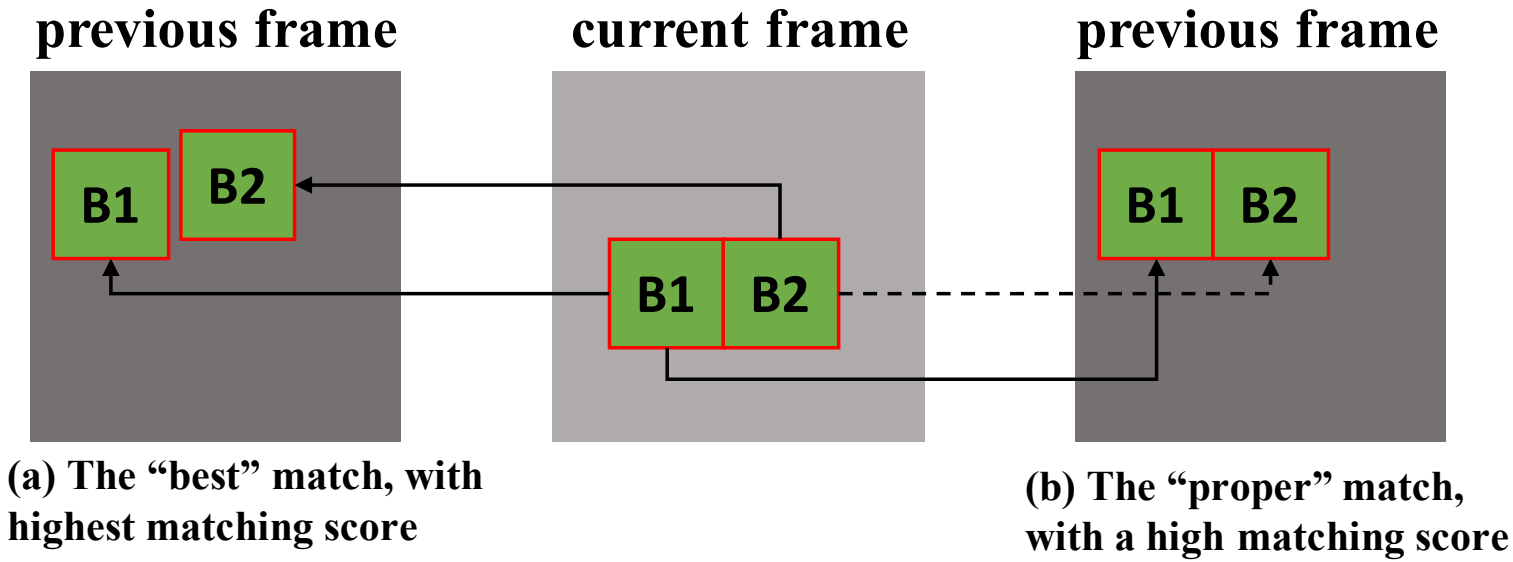
- **Cache erosion** – reusability tends to diminish at its deeper layers



# Challenges of DeepCache

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

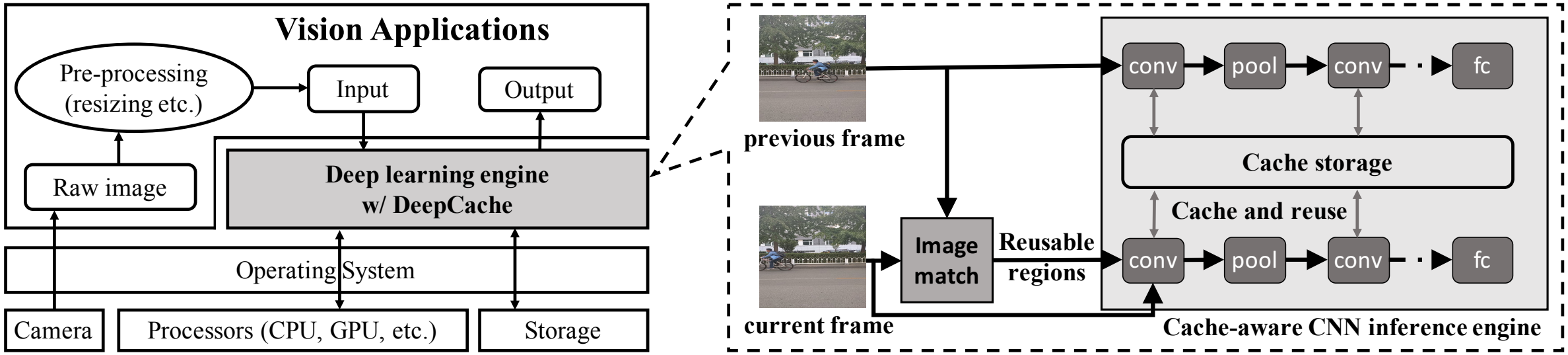
1. Merge smaller ones




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

# DeepCache Design: Overview

- **Design Principles**
  - No cloud offloading
  - No efforts from developers
  - No modification to models
- **Two modules**
  - **Image matcher**
  - **Cache-aware inference engine**

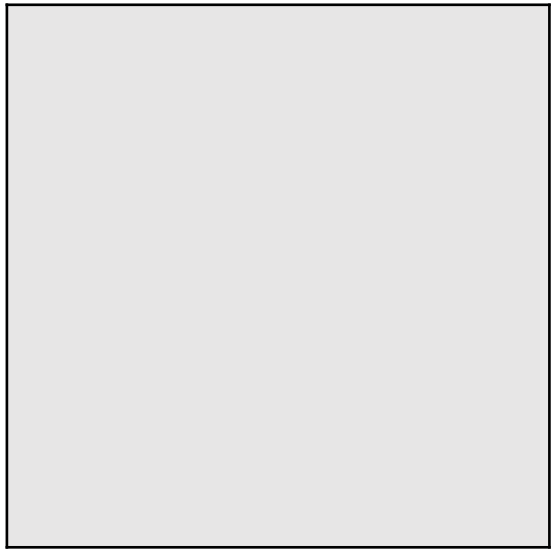


# DeepCache Design: Image Matching

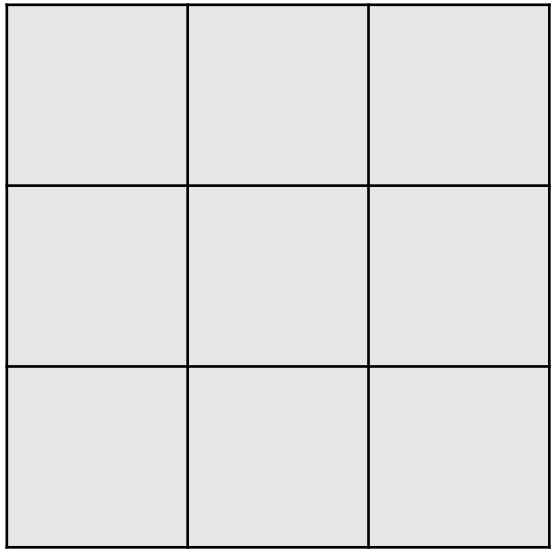
- Principles: high similarity, low overhead, and merged to big ones.
- Input: two raw images
- Output: a set of matched rectangles
  - ($x_1$, $y_1$, w, h) in current frame -> ($x_2$, $y_2$, w, h) in previous frame

# DeepCache Design: Image Matching

- Step 1: dividing the current frame into an NxN grid.
  - N is a configurable parameter (default: 10 x 10).



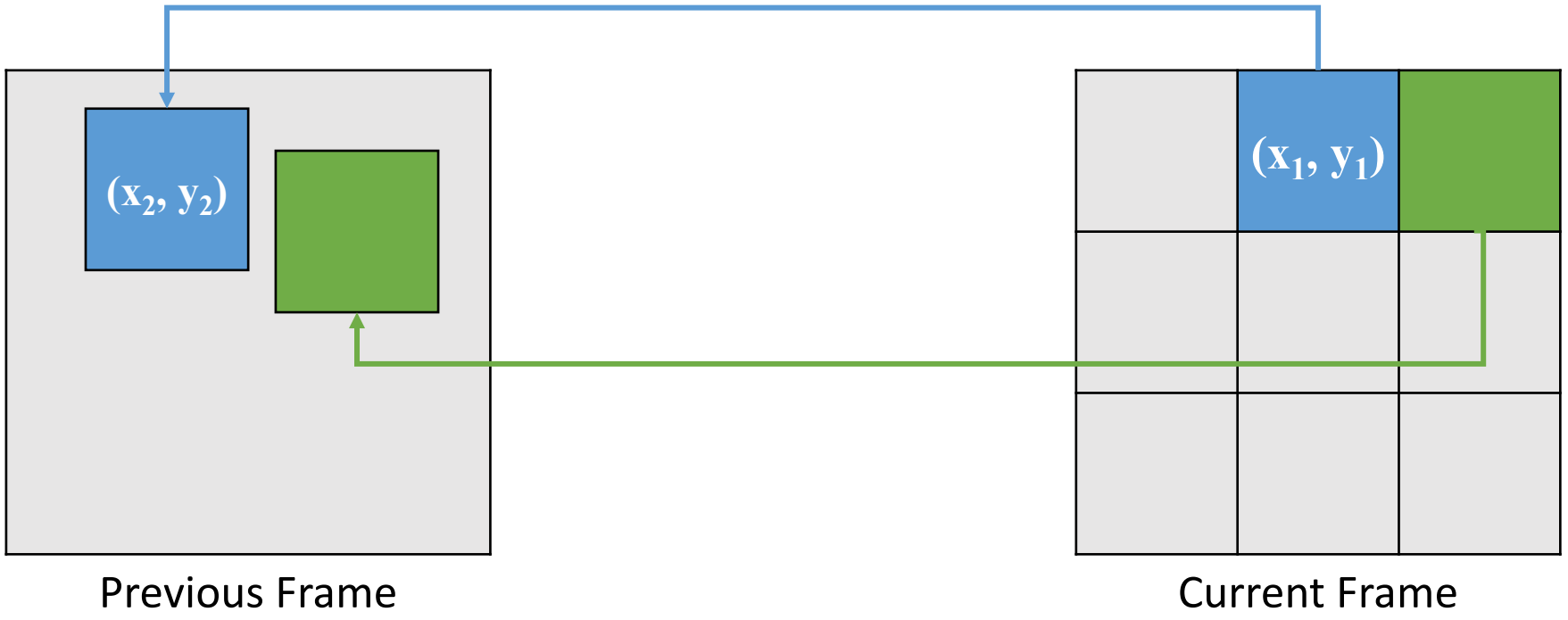
Previous Frame



Current Frame

# DeepCache Design: Image Matching

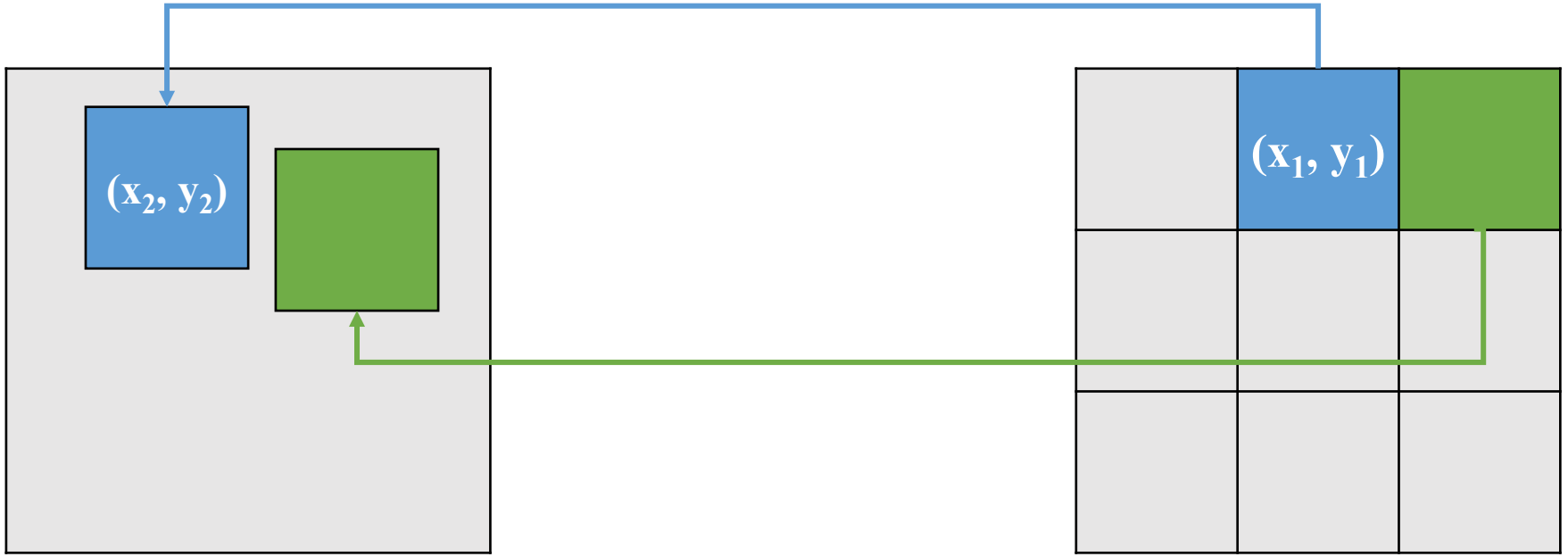
- Step 2: find the most-matched block in previous frame for each divided block
  - Motion estimation: *diamond search*



# DeepCache Design: Image Matching

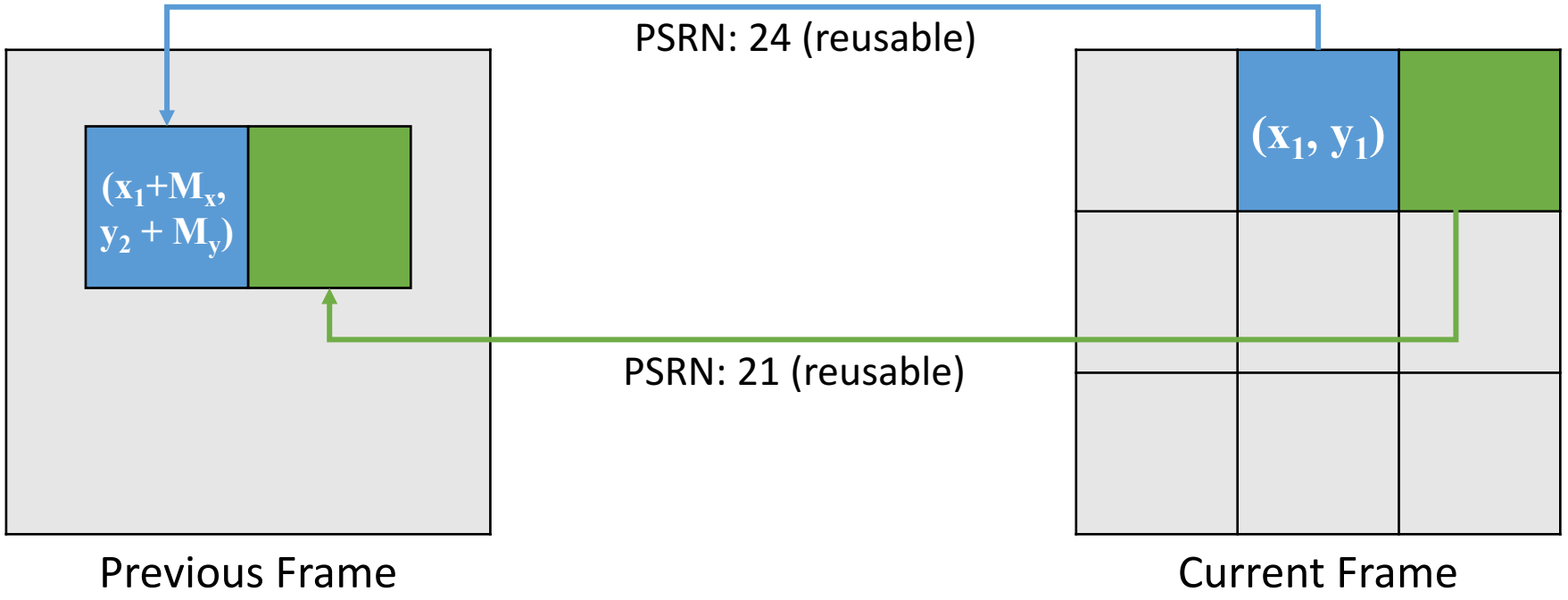
- Step 3: calculate the average block movement (offset): ($M_x$, $M_y$).
  - Filter those outliers

$$(M_x, M_y) = \left(\frac{\Sigma(x_i' - x_i)}{K}, \frac{\Sigma(y_i' - y_i)}{K}\right), \langle (x_i, y_i), (x_i', y_i') \rangle \in \mathcal{S}$$



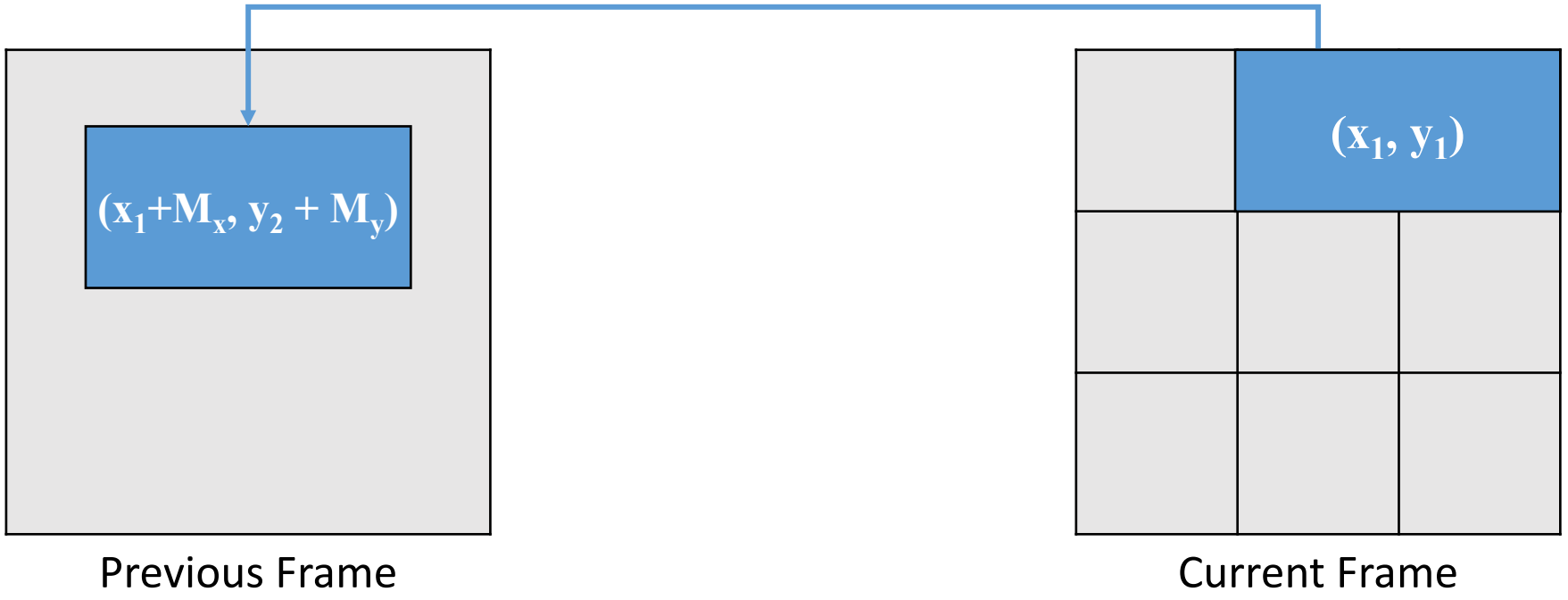
# DeepCache Design: Image Matching

- 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)



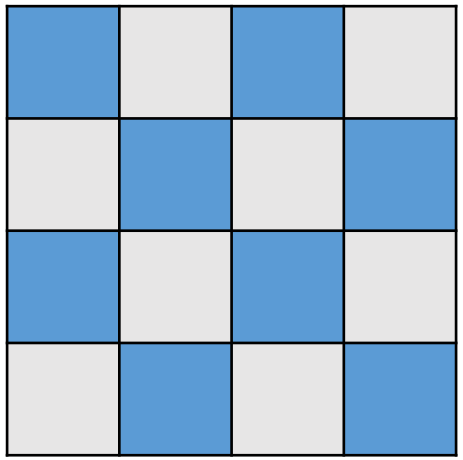
# DeepCache Design: Image Matching

- Step 5: merge blocks into larger ones if possible

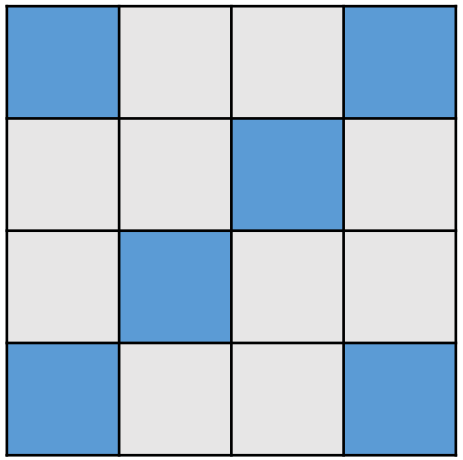


# DeepCache Design: Image Matching

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



2-skip
8/16 blocks
computed

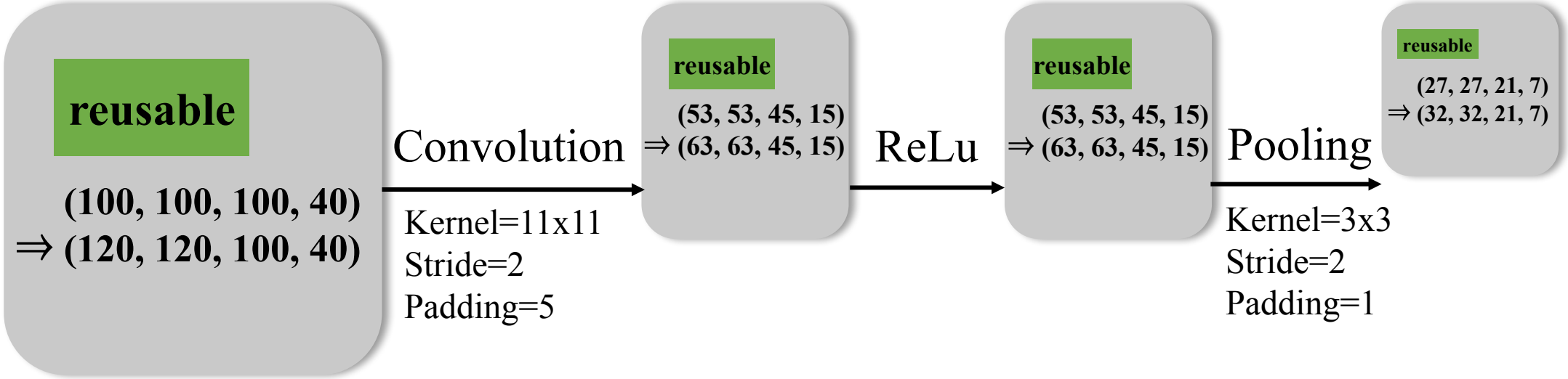


3-skip
6/16 blocks
computed

- 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.

| **Layer Type** | **Layer Parameters** | **Output($\mathcal{D}_t$)** |
|---|---|---|
| Convolution | kernel=k x k stride=s, padding=p | $x' = \lceil (x+p)/s \rceil$, $y' = \lceil (y+p)/s \rceil$<br>$w' = \lfloor (w-k)/s \rfloor$, $h' = \lfloor (h-k)/s \rfloor$ |
| Pooling | | |
| LRN [10] | radius=r | $x' = x + r$, $y' = y + r$<br>$w' = w - 2 * r$, $h' = h - 2 * r$ |
| Concat [7] | input number=$N$ | overlapped region of these $N$ rectangles |
| Fully-connect | / | $(x', y', w', h') = (0, 0, 0, 0)$ |
| Softmax [16] | | |
| Others | / | $(x', y', w', h') = (x, y, w, h)$ |

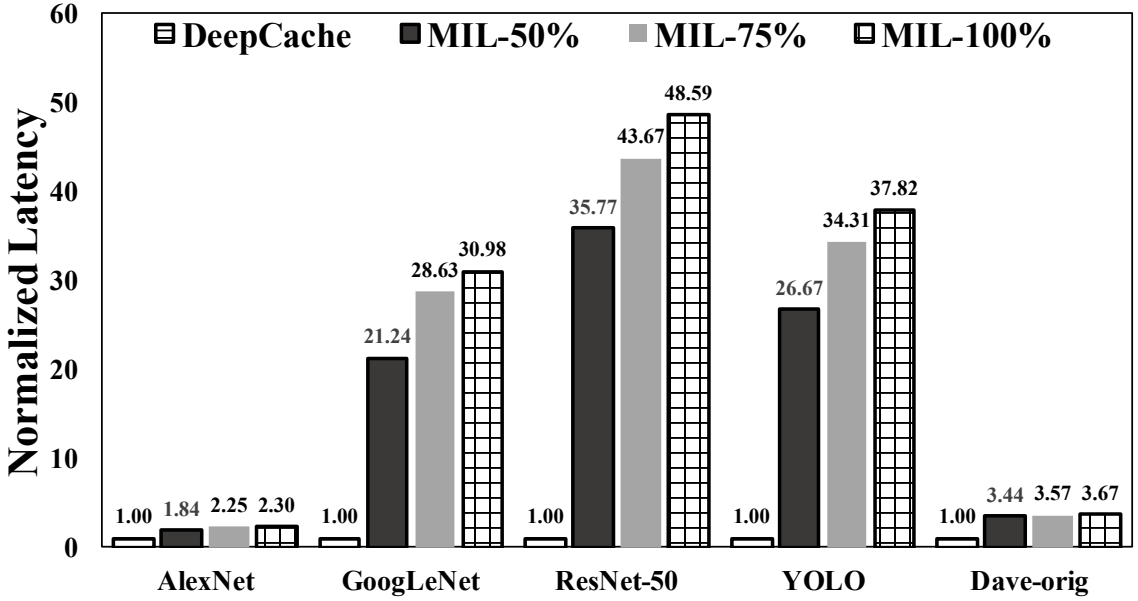
partial erosion

full erosion

no erosion

# DeepCache Design: Cache-aware CNN Inference

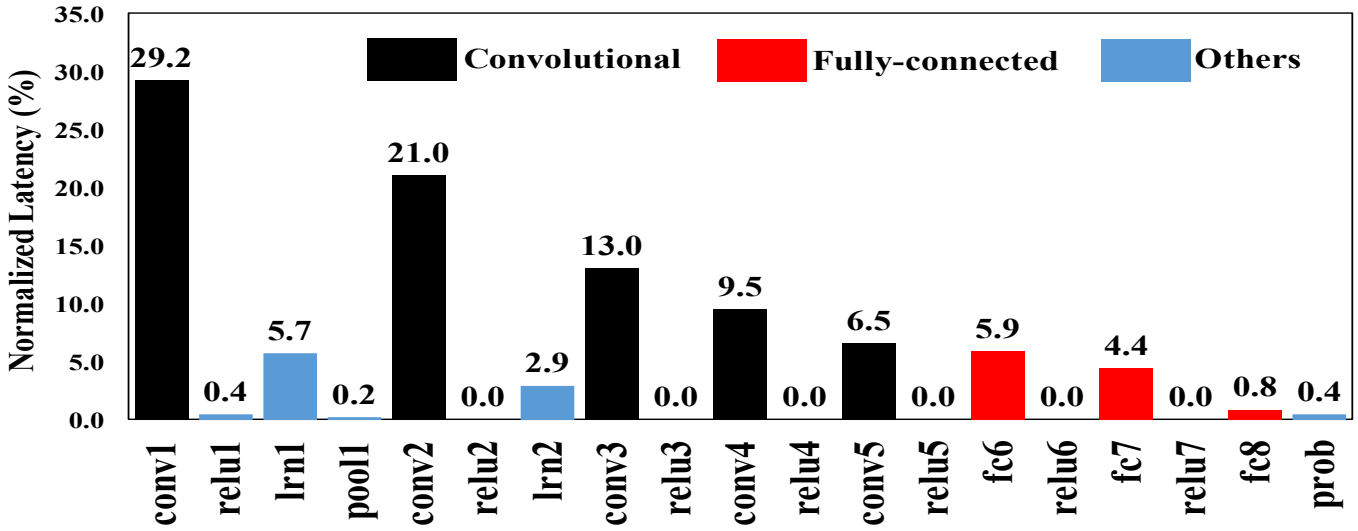
- **Why Propagation**? Why not match the input of each layer?
  - Low return: feature maps are high dimensional data, difficult to interpret.
  - High cost: matching feature map requires a lot of computations (40× compared to propagation for ResNet).




- ***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.




*Convolution is often the dominant layer (> 80% overall computations)*

- 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

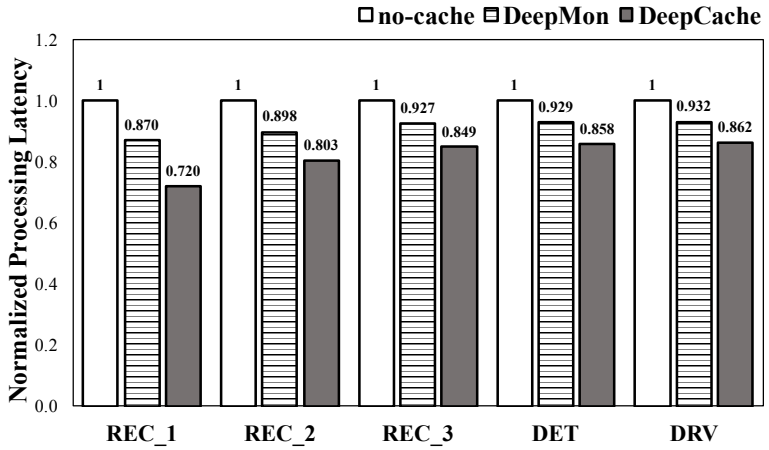
| Application | Model Name | Architecture | # of Conv | Model Output | Dataset |
|---|---|---|---|---|---|
| Activity recognition | REC_1 | AlexNet [43] | 5 | human activity type | UCF101 [57] |
| | REC_2 | GoogLeNet [58] | 57 | | |
| | REC_3 | ResNet-50 [35] | 53 | | |
| Object detection | DET | YOLO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

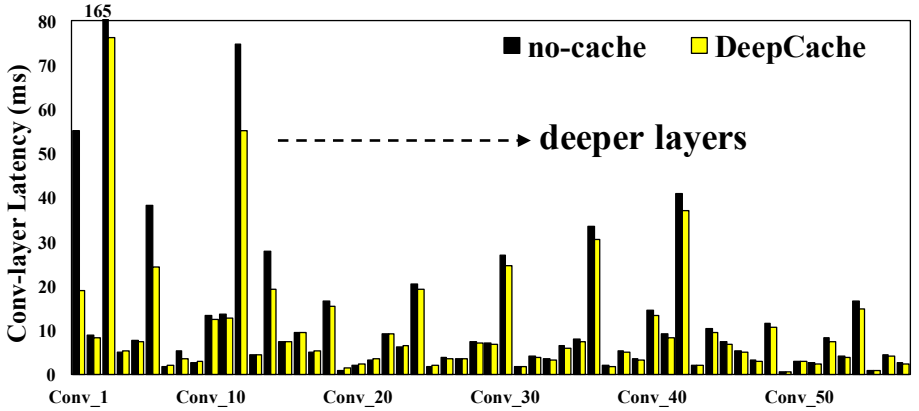
- Platform: Nexus 6, Android 6.0
- Alternative
  - *ncnn* without cache
  - Coarse-grained cache used in [1]DeepMon

[1] Huynh Nguyen Loc, Youngki Lee, and Rajesh Krishna Balan. 2017. *DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications*. In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)

# Evaluation – Execution Speedup

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

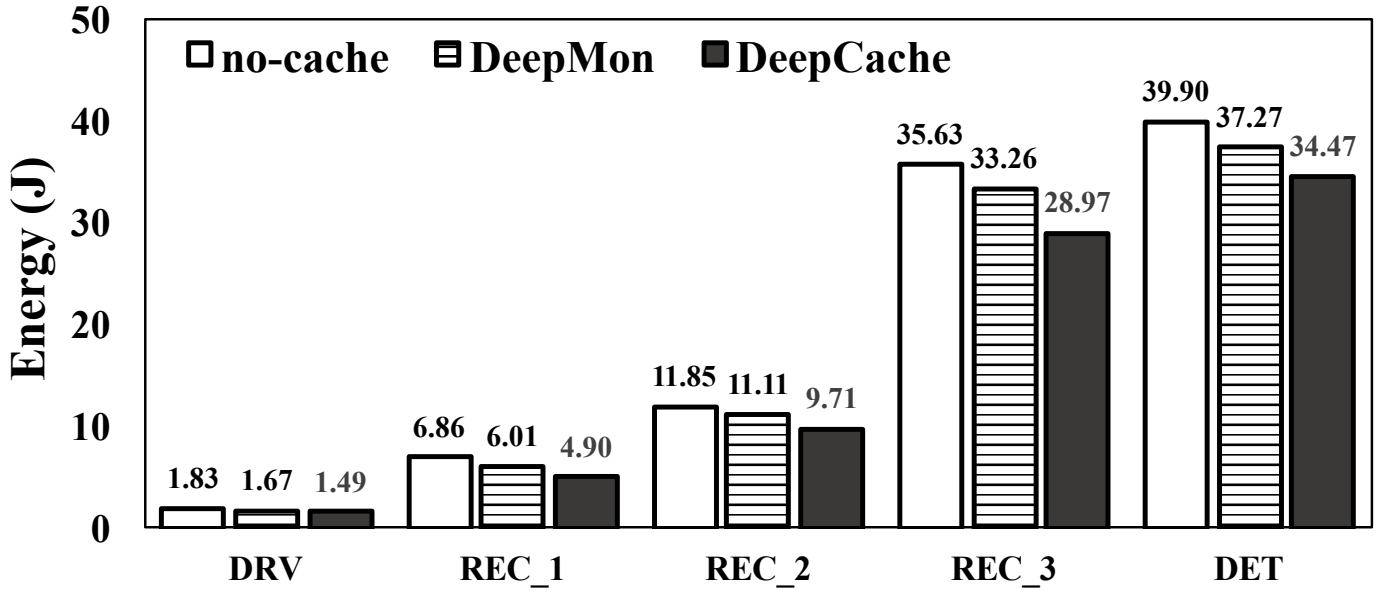



| Application | Model Name | Architecture | # of Conv | Model Output | Dataset |
|---|---|---|---|---|---|
| Activity recognition | REC_1 | AlexNet [43] | 5 | human activity type | UCF101 [57] |
| | REC_2 | GoogLeNet [58] | 57 | | |
| | REC_3 | ResNet-50 [35] | 53 | | |
| Object detection | DET | YOLO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

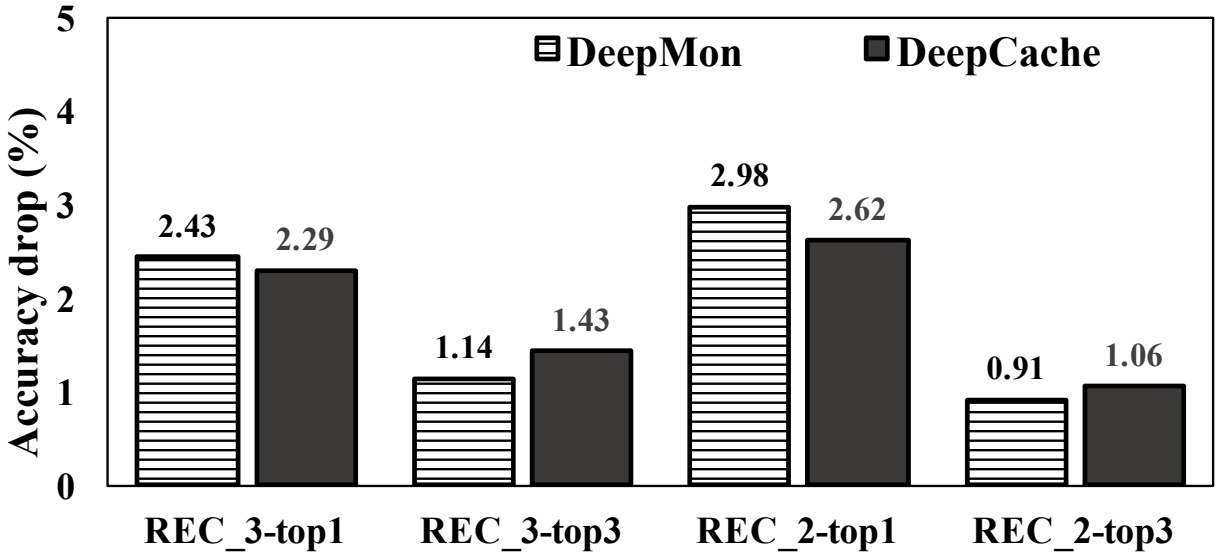
# Evaluation – Energy Saving

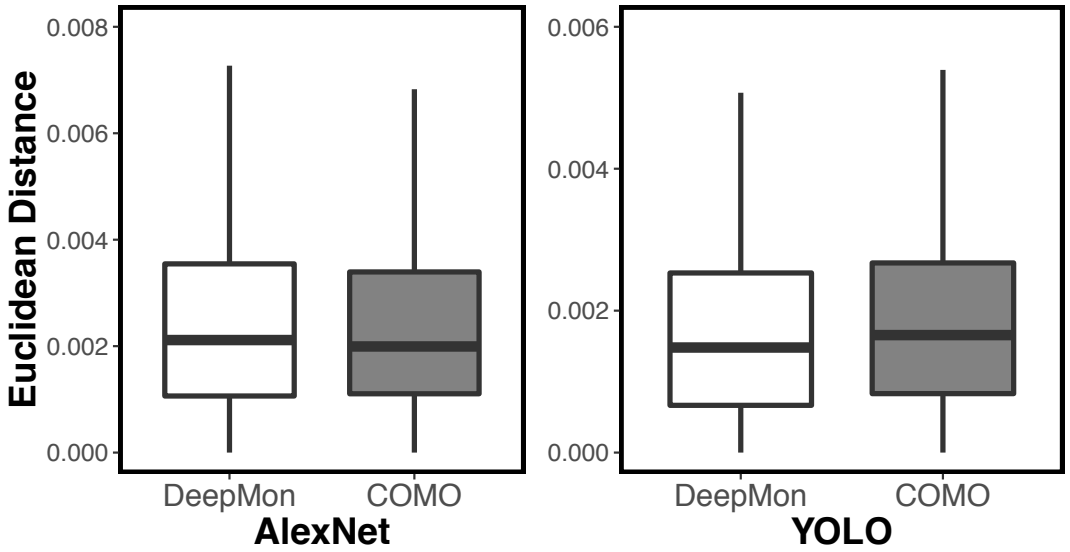
- DeepCache can save around 20% energy consumption of processing same number of images.
  - The energy saved by using DeepCache to process 10 images (ResNet) is equal to 40 seconds of playing video on mobile devices.



# Evaluation – Accuracy

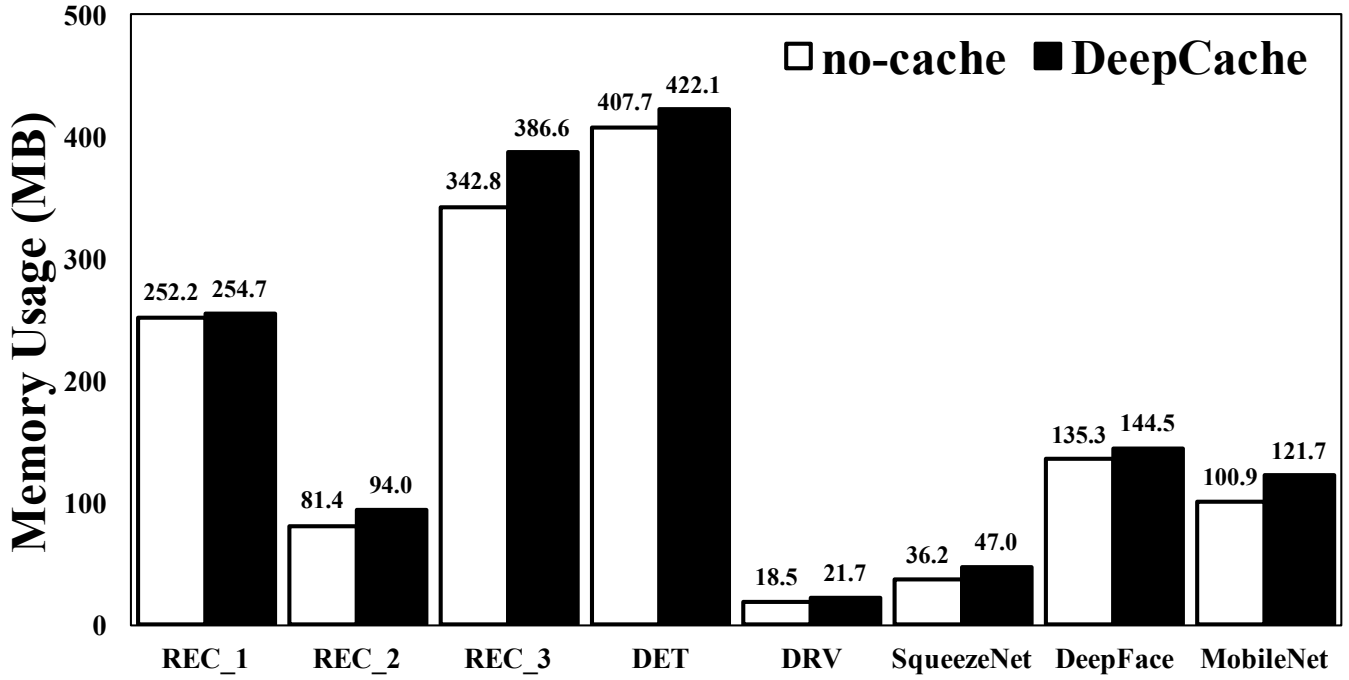
- DeepCache is able to keep the accuracy
  - 2% on average, similar to DeepMon



# 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



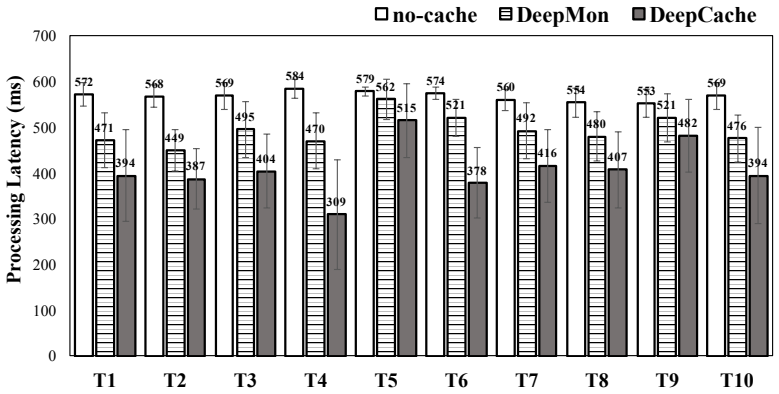
# Conclusion

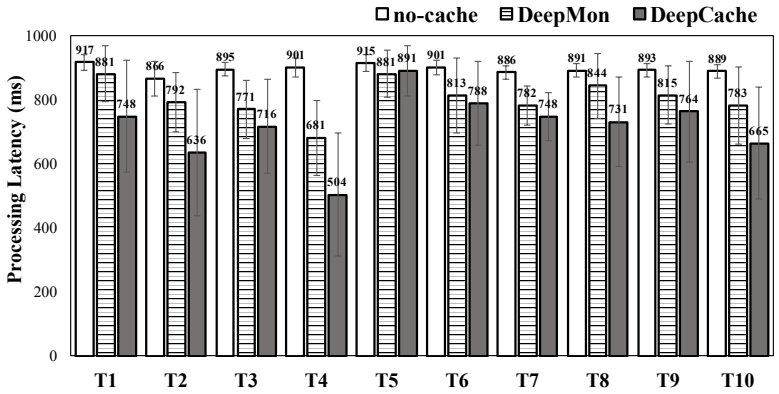
- DeepCache: cache design for mobile deep vision
  - Image matching on raw images
  - Cache/reuse in inference engine
- Evaluation
  - ~20% execution speedup and energy savings
  - Little accuracy loss

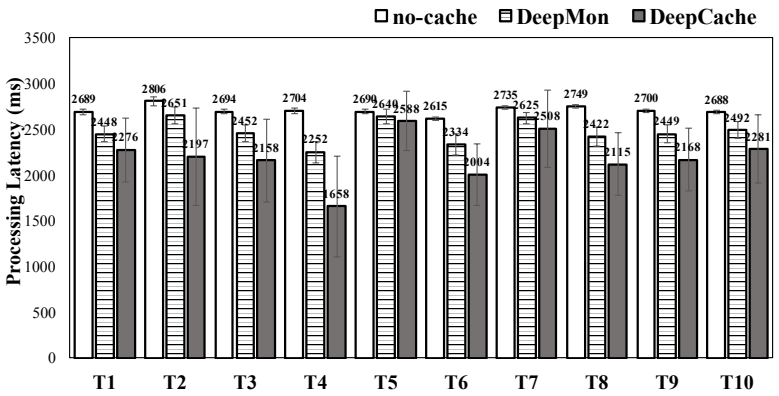
Thank you for attention!

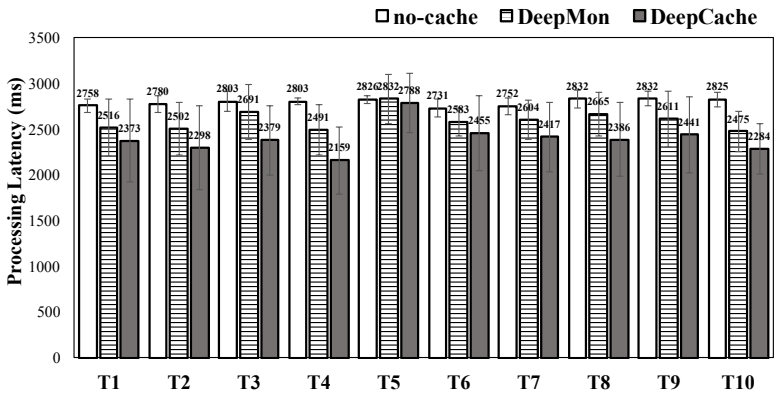
# Evaluation – Execution Speedup

- 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