Boosting Mobile CNN Inference through Semantic Memory

Yun Li∗
University of Science and Technology of China
yli001@mail.ustc.edu.cn

Chen Zhang†
Damo Academy, Alibaba Group
mingchong.zc@alibaba-inc.com

Shihao Han∗
Rose-Hulman Institute of Technology
hans3@rose-hulman.edu

Li Lyna Zhang
Microsoft Research
lzhani@microsoft.com

Baoqun Yin†
University of Science and Technology of China
bqyin@ustc.edu.cn

Yunxin Liu
Institute for AI Industry Research (AIR), Tsinghua University
liuyunxin@air.tsinghua.edu.cn

Mengwei Xu
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
mwx@bupt.edu.cn

ABSTRACT
Human brains are known to be capable of speeding up visual recognition of repeatedly presented objects through faster memory encoding and accessing procedures on activated neurons. For the first time, we borrow and distill such a capability into a semantic memory design, namely SMTM, to improve on-device CNN inference. SMTM employs a hierarchical memory architecture to leverage the long-tail distribution of objects of interest, and further incorporates several novel techniques to put it into effects: (1) it encodes high-dimensional feature maps into low-dimensional, semantic vectors for low-cost yet accurate cache and lookup; (2) it uses a novel metric in determining the exit timing considering different layers’ inherent characteristics; (3) it adaptively adjusts the cache size and semantic vectors to fit the scene dynamics.

SMTM is prototyped on commodity CNN engine and runs on both mobile CPU and GPU. Extensive experiments on large-scale datasets and models show that SMTM can significantly speed up the model inference over standard approach (up to 2×) and prior cache designs (up to 1.5×), with acceptable accuracy loss.

KEYWORDS
neural networks, semantic memory, mobile CNN inference

1 INTRODUCTION
The recent advances of Convolutional Neural Networks (CNNs) have catalyzed many emerging mobile vision tasks, including but not limited to augmented reality, face recognition, activity recognition, etc [20, 54, 57]. A notable trend is on-device CNN inference as against cloud offloading due to the tight delay constraint and data privacy concerns [4]. For instance, the Android applications empowered by on-device deep learning have increased by 27% within only a quarter in 2018, where CNNs dominate the use cases (>85%) [49].

The key challenge to fit CNN to resource-constrained mobile devices is its high computation load, especially in continuous vision tasks where predictions are performed on a stream of image frames. A unique opportunity to accelerate continuous vision inference resides in its high temporal locality: recently seen objects are more likely to appear in the next few frames, and the frequency of object occurrence in the vision streams typically follows a long-tail distribution. Those observations straightforwardly motivate a CNN system to "memorize" the recent inference results and directly omit the future inference if similar inputs are observed.

Indeed, such a memory mechanism also exists in human brains, from which neural network borrows its spirits exactly. Human brain leverages temporal redundancy with priming effect, a psychology phenomenon whereby exposure to one stimulus improves a response to a subsequent stimulus, without conscious guidance or intention [5, 46]. Dated back to the 70s, biologic experiments [38] already show that human brain speeds up the recognition of repeatedly presented objects due to faster memory encoding and accessing procedures on activated neurons. The cognitive neuroscience research [14] also reveals that the priming effect is related to the long- and short-term memory of human brains: recent and just seen objects are stored in fast memory (short memory) and faster to be recognized than infrequently seen objects. In a nutshell, human brains seem to be born with a kind of semantic cache mechanism.

A few recent studies have tried to exploit the opportunity of temporal redundancy. However, unlike human brains that focus on semantic, high-level visual information, those systems only consider low-level visual information (either image pixels or blocks) by matching the input images [22] or intermediate feature maps (activations) [51]. As a result, the memory efficiency can be easily compromised by scene variation, e.g., object movement or light...
The 'priming effect' is a fundamental cognitive phenomenon and is born with the implicit memory in human brains [14]. It refers to the changes in processing speed, bias or accuracy of one stimulus, as the data distribution of the scene is not known in advance, we prototype a cache replacement policy that takes frequency and recency of the observed data to predict the recurrence probability of each class in the future. Moreover, we propose an adaptive cache size and an adaptive semantic center to increase cache hit ratio and recognition accuracy under various and high-moving scenarios.

We summarized our major contributions.

- Semantic memory, a novel cache mechanism borrowed from neuroscience research, to accelerate on-device CNN inference.
- Three concrete techniques to take the semantic memory into effects: an accurate yet low-cost memory encoder, an early exit method, and an adaptive priming memory policy.
- A prototype of SMTM on commodity CNN engine and extensive experiments showing its effectiveness.

2 INSPIRATION AND CHALLENGES

In this section, we present the inspiration for our ideas and challenges to cache semantic for mobile vision.

2.1 Lessons from cognitive neuroscience

Convolutional neural network is initially designed to imitate optic neurons of mammals due to high recognition accuracy on processing computer vision data [27]. However, today’s CNN models are of high computation complexity and energy intensity, putting a lot pressure on resource-constrained mobile and wearable devices. The human brain is much more efficient, which achieves high intelligence within a small power envelope.

The 'priming effect' is a fundamental cognitive phenomenon and is born with the implicit memory in human brains [14]. It refers to the changes in processing speed, bias or accuracy of one stimulus,
Although the objective of caching and reusing semantics to accelerate CNN execution is intuitive, we notice there are several major obstacles to build an effective memory mechanism like human brains.

- **Efficient memory encoding against CNN models’ over-parameterization** (Section 4). The underlying representational power of today’s CNN models comes from the huge parameter space which results in an extremely large volume of intermediate data, i.e., feature maps, generated in hidden layers. Processing these feature maps requires a large memory footprint and high computation cost, which impedes fast memory encoding and accessing. An accurate yet low-cost memory encoding is desired to represent the high-level vision semantics from CNN layers’ feature maps. SMTM introduces GAP function as the encoding tool to generate memory encoding from multi-dimensional feature maps.

- **Obtaining speedup by high-level vision semantics** (Section 5). Previous memory designs [22, 51] mainly cache the hot-spot by using low-level vision information, e.g., measuring pixel-level similarities. However, human brain makes recognition of an object by its high-level features instead of pixels’ digital values. Accelerating CNN inference by leveraging the high-level semantics requires a co-design of the proposed memory encoding and CNN’s execution flow. SMTM demonstrates the feasibility of exiting inference early on intermediate CNN layers by using high-level semantics.

- **Battling dynamics on scenario variation** (Section 6). On mobile or wearable devices, the scene may change drastically from time to time with the movement of the user/camera. Such a high dynamics raises two issues. First, the data distribution and the scene complexity is not known in advance. For example, an auto-driving car mainly needs to recognize cars on highways but also has to deal with a much larger number of object classes on downtown streets, e.g., traffic lights, pedestrians, stop signs, etc. Second, the characteristics of real scenario data may differ from the training set on which the model is trained. This data variation may result in accuracy loss with more real-scenario data accumulated. To tackle the above challenges, we propose two techniques, 1) an adaptive cache size to adjust for different scenarios, and 2) an online method to update semantic vectors with the input image.

## 2.2 Challenges

2.2 Challenges

Although the objective of caching and reusing semantics to accelerate CNN execution is intuitive, we notice there are several major obstacles to build an effective memory mechanism like human brains.

- Efficient memory encoding against CNN models’ over-parameterization (Section 4). The underlying representational power of today’s CNN models comes from the huge parameter space which results in an extremely large volume of intermediate data, i.e., feature maps, generated in hidden layers. Processing these feature maps requires a large memory footprint and high computation cost, which impedes fast memory encoding and accessing. An accurate yet low-cost memory encoding is desired to represent the high-level vision semantics from CNN layers’ feature maps. SMTM introduces GAP function as the encoding tool to generate memory encoding from multi-dimensional feature maps.

- Obtaining speedup by high-level vision semantics (Section 5). Previous memory designs [22, 51] mainly cache the hot-spot by using low-level vision information, e.g., measuring pixel-level similarities. However, human brain makes recognition of an object by its high-level features instead of pixels’ digital values. Accelerating CNN inference by leveraging the high-level semantics requires a co-design of the proposed memory encoding and CNN’s execution flow. SMTM demonstrates the feasibility of exiting inference early on intermediate CNN layers by using high-level semantics.

- Battling dynamics on scenario variation (Section 6). On mobile or wearable devices, the scene may change drastically from time to time with the movement of the user/camera. Such a high dynamics raises two issues. First, the data distribution and the scene complexity is not known in advance. For example, an auto-driving car mainly needs to recognize cars on highways but also has to deal with a much larger number of object classes on downtown streets, e.g., traffic lights, pedestrians, stop signs, etc. Second, the characteristics of real scenario data may differ from the training set on which the model is trained. This data variation may result in accuracy loss with more real-scenario data accumulated. To tackle the above challenges, we propose two techniques, 1) an adaptive cache size to adjust for different scenarios, and 2) an online method to update semantic vectors with the input image.

## 3 SYSTEM DESIGN OVERVIEW

SMTM speeds up CNN inference by skipping some layers’ execution according to the cached (activated) memory of frequently and recently-seen objects. The key advantages of SMTM include: 1) Small memory footprint and low memory lookup cost. Instead of directly storing the multi-dimensional feature maps, SMTM only uses a set of vectors as memory encoding to encode high-level vision information for each category. The memory encoding has a reduced dimension and thus takes much less memory and memory lookup cost. 2) Enjoy pervasive AI hardware acceleration. SMTM is capable to directly skip CNN layers. In contrast to pixel/region reuse [22, 51], SMTM does not require any modification to the original convolution operator, and can directly use existing mobile AI hardware for acceleration, e.g SIMD units or GPUs. 3) Soft constraints on the matching object. Different from low-level vision (pixel or region) reuse, SMTM explores temporal redundancy according to high-level vision information. The reused objects only need to be of the same category that has high-level feature similarity, but not have to follow pixel-level similarity constraints. This helps to discover more temporal redundancy and enables more acceleration chances. 4) Robust on scene variations. SMTM propose an adaptive policy to adjust the priming memory mechanism for different types of variations in a mobile device.

Workflow. Figure 2 shows the overall workflow of SMTM. SMTM introduces a global memory and a fast memory to improve the process of traditional CNN inference. First, SMTM employs a global memory to cache the frequency and timestamp of all classes, as well as the feature expression of each class extracted in the training set. Second, SMTM uses a fast memory to cache a few hot-spot classes and their features for fast matching. The caching replacement policy...
predicts the possibility of objects’ occurrence in the mobile video streams according to two observations: 1) a long-tail distribution: some objects are much more frequently seen than others. 2) temporal locality: a recently-seen object is more likely to appear in the next few frames. During the CNN inference process, SMTM extracts the intermediate feature per layer and matches them with the cached features in fast memory. Once matched, SMTM skips the rest of the layers and directly outputs the final results. At last, SMTM updates the frequency table and time-stamp table, which will be used in the memory replacement policy periodically.

SMTM proposes the following technologies to solve the challenges mentioned before.

**Semantic memory encoding: perform memory encoding and lookup. (Section 4)** SMTM encodes the large volume of feature maps to low-dimensional semantic vectors in memory. It does so based on the global average pooling (GAP) [31], a widely used function in person Re-ID tasks [45, 52] as a dimension reduction and high-level feature extraction operation to perform person matching. We present a detailed analysis of semantic vectors’ grouping performance that is used to distinguish different classes’ categories.

**Early exit: obtaining speedup with a novel metric. (Section 5)** As CNN performs inference layer by layer, SMTM calculates the semantic vector on each layer’s output feature map. Then, the semantic vector works as the ‘keys’ to lookup corresponding objects’ categories as ‘values’ by measuring cosine similarities. To increase the robustness of prediction accuracy, SMTM proposes the cross-layer cumulative similarity to determine the exiting timing by combining the confidences of multiple layers.

**Adaptive Priming Memory: cache and update the semantics of mobile video frames. (Section 6)** To reuse semantics efficiently to deal with the scene variation in mobile video, SMTM sets a memory replacement policy in the global memory, which maintains a frequency table to record the time of each object presented in the history, and a time-stamp table to record the consecutive non-appearing frames of each object recently. Then, the cache replacement policy will calculate a score based on the current frequency table and the time-stamp table to select the object classes with the highest recurrence probability, and cache them in the fast memory. To overcome the high dynamic scenarios in mobile vision video, SMTM introduces the adaptive cache size and the adaptive semantic centers. A probability estimation method is adopted to tune the cache size in fast memory based on the current frequency table and time-stamp table. To make the semantic centers constantly adapt to test scenarios, SMTM gradually update the semantic centers by accumulating the semantic vector in a weighted average manner.

## 4 SEMANTIC MEMORY ENCODING

An efficient memory encoding is a prerequisite because the large volume of feature maps introduces high computation cost as well as large memory footprint overhead. It finally impedes fast memory encoding and accessing for a priming effect.

Besides being fast, the memory encoding must accurately capture the key features of the corresponding objects so that the image semantics can differentiate different classes. We use separability to evaluate the memory encoding on classifying different memories. During inference runtime, we adopt the metric learning method cosine similarity [35] to measure the separability between memories and the semantics of the new input layer by layer.

Next, we first introduce semantic vector that we use to encode memory. Then, we evaluate the separability within semantic vectors of each layer.

### 4.1 Semantic vectors

SMTM memorizes intermediate data during CNN inference. An early exit happens on the condition that some layer’s feature map matches with the memory of some previous inputs. To efficiently memorize the intermediate data across CNN layers, we propose semantic vector that is retrieved by applying a global average pooling (GAP) function [31] on feature maps. The global average pooling takes the average of each feature map and outputs a result vector. As shown in Figure 3, the GAP has applied to each layer’s (modules’) output feature maps.

Widely used in person Re-ID tasks [45, 52], global average pooling (GAP) serves as a dimension reduction and key feature extraction to perform person matching by measuring vector distances. Similar to person Re-ID tasks, we use semantic vectors as IDs (or keys) for object classes. By measuring the similarity between semantic vectors, we can establish the mapping between each individual’s semantic vector and object classes.

Semantic vectors have preferred characteristics of small memory footprint and low computational cost because it greatly reduces raw feature maps’ dimensions. Given a feature map size of $C \times H \times W$, the semantic vector is only $C$, where $(C, H, W)$ stands for channels, row, and column, respectively. Despite that, the semantic vectors also have good separability, which makes it easier to differentiate objects of various classes from semantic vectors. We provide a detailed analysis in the following subsection.

### 4.2 Rationales

Intra-class distance and inter-class distance [39] are two key indicators to measure the semantic vectors’ performance on clustering data. When the intra-class distance is smaller than the inter-class distance, we can distinguish different targets well. To analyze its separability of different objects, we sample three hidden layers’ semantic vectors in the VGG-16 model and visualize them in Figure 4. Since semantic vectors are still multi-dimensional, we use t-SNE method [33] for visualization. In each subfigure of Figure 4, the bigger labels mean the semantic centers of each object class, and the smaller labels denote the semantic vector of test samples. We can draw two conclusions from Figure 4. First, semantic vectors...
Figure 4: Visualized separability of semantic vectors for different VGG16 layers, showing that going deeper the semantic vectors can be more accurately separated.

5  EARLY EXIT

SMTM uses the semantic vectors to capture the repeatedly seen objects and save computation workload by the early exit. During the CNN inference, we first encode the semantic vectors layer by layer and then match them with the semantic centers of objects in memory. If there is enough confidence about the matching results, the CNN inference will exit early and the rest layers are skipped directly. The semantic centers are initialized with a grouping center (the average of semantic vectors) on the training dataset and will be updated during runtime, which will be discussed later.

Based on the observation from Figure 4, the semantic vector’s separability in shallow layers is not as strong or stable as the deeper layers. To exit the inference as early as possible while ensuring inference accuracy, we adopt the cross-layer cumulative similarity to evaluate memory matching results during the memory lookup period. Overall, the memory lookup can be divided into three steps.

**Step 1.** For the current input frame (image), we adopt global average pooling to encode the memory and generate semantic vectors layer by layer during the CNN inference.

**Step 2.** At each layer, we leverage cosine similarity [35] to lookup the most matched semantic centers in memory. The matching level is represented by similarity $s^l_j$ in Eq.1. To improve the robustness to distinguish all objects in a single layer, we introduce the cross-layer cumulative similarity to evaluate memory matching result:

$$SA^l_j = \sum_{l_0=1}^{l} s^l_j \times weight_{l_0}, j \in [1, n],$$

where $SA^l_j$ is the similarity accumulation result between the semantic vector of the current frame and semantic center of the object $j$ in memory from layer 1 to layer $l$, $weight_{l_0}$ is the weight of the results in the $l_0$-th layer. Considering the separability will become more and more obvious as the CNN layer going deep, the preceding equation requires a sequence of increasing weight values. To this end, we adopt an exponential ($weight_{l_0} = 2^{l_0-1}$) weighted decay. As this exponential function has a useful characteristic ($\sum_{l_0=1}^{l} 2^{l_0-1} = 2^{l-1} - 1 = weight_{l_1} - 1$), making the weight of current layer $l$ and the cumulative weights of previous shallow layers almost equal. This weighting method not only ensures that
we propose to tune the fast memory size adaptively according to the frequency table. Suppose the next video frame contains the object, then the probability of event \( A = \{ \theta \in \Psi \} \) can be a subset of \( \Phi \times \{0,1\} \), in which \( W \) is the size of observation time window. According to the values in the time-stamp table, we decay the memory (the effect of the corresponding frequency on the cache updating) every consecutive \( W \) frames.

The replacement policy takes the Top-k highest score that are calculated by the following equation to select the objects from the global memory and cache them in the fast memory,

\[
Score_i = FT_i \cdot \left( \frac{TS_i}{W} \right), \quad i \in [1, n],
\]

where \( FT_i \) is the frequency of object \( i \) in the frequency table, \( TS_i \) is the consecutive non-appearing frames of object \( i \) in the time-stamp table.

This equation lets fast memory cache the constantly-often-seen and the most-recently-seen objects. For the frequently-seen but NOT recently-seen objects, its overall score will be degraded due to a high decay ratio by the time-stamp table, and vice versa. For objects that are NOT frequently-seen and NOT recently-seen, it has the least possibility to be cached. With this policy, SMTM can enjoy inference speedup by keeping the hottest classes in fast memory.

6.2 Adaptive cache size

Due to the mobile scene’s drastic data variation, the number of hot-spot classes may vary a lot under different scenes. Using a large fast memory (cache) for a simple scenario causes overhead on memory retrieving. Thus, SMTM adopts a probability estimation method to figure out the optimal cache size.

Considering that the frequency table and the time-stamp table can reflect the frequency and the recency of the objects presented in the video stream. Thereby, SMTM maintains a frequency table and a time-stamp table for cache replacement.

The frequency table keeps a record of the number of times that each object class presented in history. It is initialized as zeros and updated by every inference output during runtime. A class with a high score in the frequency table means it has a high probability of witnessing throughout the entire ‘history’, such as ‘cars’ for downtown street cameras. Thus, the corresponding class should be given higher priority when promoting it from global memory to fast memory.

The time-stamp table keeps a record of the recency of each object class. The intuition is that the most recently seen objects are likely to appear again in the next few frames. The time-stamp table works like a forgetting mechanism [12], where all object classes decay with time. Upon every inference complete, all other classes that have not been witnessed on a time interval will be decayed by a certain ratio so that most recently-seen have the highest score at present. In our experiment, the forgetting mechanism is defined as \( \hat{\gamma}_i = \gamma_i \times (0.25)^{\frac{TS_i}{W}} \), in which \( W \) is the size of observation time window. According to the values in the time-stamp table, we decay the memory (the effect of the corresponding frequency on the cache updating) every consecutive \( W \) frames.

The replacement policy takes the Top-k highest score that are calculated by the following equation to select the objects from the global memory and cache them in the fast memory,

\[
Score_i = FT_i \cdot \left( \frac{TS_i}{W} \right), \quad i \in [1, n],
\]

where \( FT_i \) is the frequency of object \( i \) in the frequency table, \( TS_i \) is the consecutive non-appearing frames of object \( i \) in the time-stamp table.

This equation lets fast memory cache the constantly-often-seen and the most-recently-seen objects. For the frequently-seen but NOT recently-seen objects, its overall score will be degraded due to a high decay ratio by the time-stamp table, and vice versa. For objects that are NOT frequently-seen and NOT recently-seen, it has the least possibility to be cached. With this policy, SMTM can enjoy inference speedup by keeping the hottest classes in fast memory.
be formulated as:

\[ P(A) = \sum_{i=1}^{k} \frac{\text{Score}_i}{\sum_{j=1}^{k} \text{Score}_j}, \]

in which \( k \) is the number of cached objects in fast memory, \( n \) is the number of objects in the global memory. According to the statistics, if \( P(A) \) can exceed the most commonly used confidence level (CL) 95% [11, 56], we can believe that the event will happen with a high probability. Therefore, based on Eq. (6), we can adaptively tune the cache size of fast memory after each inference to overhead the scene changes. The experiments in Section 8.8 have shown that such a technology can bring 21.6% hit ratio improvement.

### 6.3 Adaptive semantic centers

The semantics center works as the key for memory lookup, which is by measuring semantic vectors’ similarity to the semantic center. An improper semantic center may finally degrade memory lookup accuracy. However, training dataset’s biases against the data in the real world may generate an improper semantic center. The scene variation may also cause a drifting optimal semantic center from time to time. To this end, we propose an adaptive semantic center method so that it can be continuously updated according to the semantics extracted from the real-world data.

SMTM initially warms up the semantic centers using the training data. During runtime, we gradually update the semantic centers by accumulating the semantic vector in a weighted average manner. For the current frame, if the predicted result is object \( j \) and the CNN inference stop at layer \( l \), then the semantic centers of object \( j \) before layer \( l \) will be updated as follow:

\[
\text{SC}_j^{l-1} = \frac{\text{SC}_j^{l-1} \cdot m_j^{l-1} + \text{SV}_j^{l-1}}{m_j^{l-1} + 1}, \quad l \in [1, L],
\]

in which, \( \text{SC}_j^{l-1} \) denotes the original semantic center of object \( j \) at layer \( l \), \( \text{SC}_j^{l} \) denotes the new semantic center of object \( j \), \( \text{SV}_j^{l} \) is the encoded semantic vector of new example, \( m_j^{l-1} \) is the update times of object \( j \) at layer \( l \), which includes the update times from the training dataset and the test scenario. By doing so, the semantic centers in global memory can be adjusted incrementally after every inference to overcome the semantic dynamics brought by highly dynamic scenarios in mobile vision tasks.

### 7 IMPLEMENTATION

We prototype SMTM atop ncnn [2], an open-source deep neural network inference computing framework optimized for mobile platforms. The ncnn provides a set of APIs for easy interaction during the model forwarding, allowing extraction of intermediate layers with relatively low overhead. While SMTM currently supports mobile CPU and GPU, it can be easily extended to more device types, e.g., DSP and NN accelerators. The implementation was split into two main parts: pre-processing of model files, and runtime inference. For the pre-processing step, we develop a tool that parses a converted ncnn model into graph presentation, inserts global average pooling (GAP) layers to predefined locations, and reconnects the graph based on requirements of ncnn by inserting split layers. Overall, the implementation of SMTM contains 4200 lines of C++ codes.

While SMTM adopts the same cache/reuse strategy for different devices, e.g., CPU and GPU, we further tune the extraction module based on hardware characteristics to further reduce the cache overhead. On CPU, we make a shallow copy of the tensor on the target extraction layer and forward through a global average pooling layer to get the feature vector. On GPU, we implement a zero-copy data path based on Vulkan API [3], allowing tensor extracted from the network to be fed into our global averaging pooling layer without going through CPU memory.

Noting that, SMTM is more instrumentation-friendly as compared to prior cache mechanisms [22, 51], because those methods require to revise the neural layer implementation (kernels). For example, ncnn has more than a hundred different implementations for convolution operation, with nearly one hundred thousand lines of code. By contrast, SMTM directly skips some complete operations in CNN and does not require any modification to the convolution calculation during inference, which can be easily applied to all the existing deep learning frameworks on mobile/wearable devices.

Our prototype is fully compatible with any existing ncnn models and applications, thus incurring zero overhead to developers. Besides, we expose key parameters, e.g., \( \tau \), exposing rich accuracy-latency trade-off to developers so it can flexibly fit into task-specific requirements. The relationship between threshold, accuracy, and latency will be given in Section 8.7.

### 8 EVALUATION

In this section, we comprehensively evaluate SMTM on diverse models, datasets, and metrics. Overall, the results show that our method can outperform existing systems by a large margin.

#### 8.1 Experimental setup

**Evaluation platform.** We evaluate SMTM on a Google Pixel 4XL mobile device, which is equipped with a Qualcomm Snapdragon 855 Mobile SoC and 6GB LPDDR4x memory. Snapdragon 855 is a big.LITTLE SoC consisting of four big Cortex-A76 cores, four little Cortex-A55 cores, and an Adreno 640 GPU.

**Benchmark Datasets.** We use two large-scale datasets UCF101 [41] and long-tail CIFAR-100 [24] to evaluate the performance of SMTM. UCF101 is an action recognition dataset of realistic video actions, including 101 action categories. The dataset consists of 13,421 short videos. Following the settings of DeepCache [51], we select 10 types as a subset for evaluation: Basketball, ApplyEyeMakeup, CleanAndJerk, Billiards, BandMarching, ApplyLipstick, CliffDiving, BrushingTeeth, BlowDryHair, and BalanceBeam. FFmpeg [1] is used to extract raw frames from those YouTube videos. Finally, 70,928 raw images are used. CIFAR-100 is for object classification task, consisting of 60,000 images, and 100 object classes.

To evaluate the robustness of SMTM against scene variation and its soft constraints on the reused objects, we adopt the long-tail CIFAR-100 as an extreme scenario. It is well-known that the frequency of object occurrence in natural scenes follows a long-tail distribution [37, 61]. Therefore, following the segmentation method in [7], we split and shuffle the CIFAR-100 test dataset into 1,442 test images with long-tail distribution to simulate the object occurrence in
natural scenes. The resolution of the input images from UCF101 and CIFAR-100 are 224 × 224 and 32 × 32, respectively. In such rapidly changing scene, to avoid the effect of the high frequency value of some objects that have not appeared for a long time, we will decay the historical memory in the frequency table based on the forgetting mechanism in Section 6.1.

**Models.** To verify that SMTM is applicable to various types of CNN architecture, we use five widely-adopted network structures: AlexNet [25], GoogleNet [42], ResNet50 [17], MobileNet V2 [21] and VGG16 [40]. For action recognition, the first four models above are used, and VGG16 is adopted to the long-tail CIFAR-100.

**Evaluation metrics.** We use five metrics to comprehensively evaluate the performance of SMTM: latency reduction (Section 8.2), accuracy loss (Section 8.3), memory overhead (Section 8.4), energy saving (Section 8.5), and early exit ratio (Section 8.6).

**Alternatives.** We compare SMTM with following alternatives: no-cache-CPU, no-cache-GPU, DeepMon [22] and DeepCache [51], no-cache-CPU/GPU use mobile CPU/GPU to compute the complete CNN models without cache reuse. DeepMon and DeepCache are two state-of-the-art cache-based approaches. To make a fair comparison with DeepCache and DeepMon, we prototype SMTM using the same inference engine (ncnn) with the same configuration as DeepCache [51] without single instruction, multiple data (SIMD). For other experiments, if not otherwise specified, we prototype the framework on the ncnn with SIMD.

### 8.2 Latency reduction

In this section, we evaluate the latency reduction on action recognition and classification when applying the proposed SMTM. The latency is tested on different devices with configuration: mobile CPU without SIMD acceleration, mobile CPU with SIMD acceleration, and mobile GPU acceleration. To evaluate the performance on the entire dataset, we adopt SMTM to accelerate the CNN inference for each input frame and calculate their average processing time.

First, we evaluate the latency reduction on mobile CPU with naive ncnn configuration. To make a fair comparison with the state-of-the-arts [22, 51], we set the same ncnn configuration (CPU w/o SIMD) as DeepCache and the results are shown in Figure 6. The open-sourced implementation of DeepMon [22] and DeepCache [51] are not compatible with the two models MobileNet V2 and VGG16, so we are not able to reproduce some results. Therefore, we only compare the results on AlexNet, GoogleNet, ResNet50. It shows that SMTM achieves 37.4% latency reduction on average on three widely used CNN models AlexNet, GoogleNet, ResNet50, while DeepMon and DeepCache have only 10.5% and 20.9%. Comparing their performance on different CNN models, SMTM reduces the processing time by 1.1x-1.4x than DeepCache, and 1.3x-1.5x than DeepMon. For the model AlexNet with only 6 convolutional layers and 3 fully connected layers, SMTM can achieve 32.9% latency reduction, while for the deeper and compact network MobileNet V2, SMTM can even save 48.5% processing time. It shows that SMTM achieves better latency reduction on the deeper network, while the performance of DeepMon and DeepCache deteriorates as the models become deeper.
to 47.7%, which reveals that our method can efficiently utilize the redundancy in continuous vision. On the extreme scenario long-tail CIFAR-100, applying SMTM on VGG16 can also achieve 30.6% latency reduction on mobile GPU and 31.10% on CPU with SIMD, which shows SMTM is robust to scene changes.

### 8.3 Accuracy loss
We then investigate how much accuracy SMTM compromises in return for the latency reduction above. The accuracy drop of SMTM is shown in Figure 8. We can see that on action recognition and image classification scenarios, introducing SMTM only leads to 1.05% latency loss on average on the five CNN models. Compared to DeepMon and DeepCache on UCF101, Semantic achieves much lower accuracy loss on Alexnet and GoogleNet. In detail, the maximum accuracy loss of SMTM on the 5 CNN models does not exceed 2.5%. In particular, for the famous GoogleNet and ResNet50 on UCF101, SMTM achieves up to 40.8% latency reduction with only 0.1% accuracy loss, which is negligible. This is because that we have designed a similarity accumulation mechanism that makes the final decision based on the matching on the whole path instead of a single layer, thus minimizing the impact of some layers’ wrongly semantic matching for the final decision.

### 8.4 Memory overhead
We also evaluate the memory overhead introduced by SMTM. The results are shown in Figure 9. As observed, the memory overhead occurred by Semantic ranges from 0.5MB to 5.3MB on the five general CNN models, with an average of 2MB, which is only 10% of the average overhead brought by the state-of-the-art method DeepCache [51]. This overhead is quite trivial for the equipped large size of memory in nowadays mobile devices, e.g., 6GB in Google Pixel 4XL. The reason for the above performance is that, unlike previous methods which store the expensive raw images and feature maps, SMTM only need to cache the semantic centers of objects, which are composed of low-dimensional vectors.

### 8.5 Energy saving
Next, we investigate the energy consumption of SMTM across all the selected test benchmarks. The energy consumption is measured via on-device PMIC (power management integrated circuit). The PMIC reports the mobile devices’ current and voltage readings at around 800Hz. A single inference is consists of data loading, data prepossessing, inference, and collecting benchmark data. As the inference time is too short, it’s difficult to capture enough data to get a valid inference time energy. Therefore, we force the device in an infinite inference loop and measuring the average voltage and current to get the power. Then, we integrate the power with the inference time collected to get the energy reports. Finally, the energy-saving ratio at different devices (CPU and GPU) are shown in Figure 10. It shows that SMTM can achieve 35.40% energy saving on average on mobile CPU and 30.60% on average on mobile GPU. The maximum energy saving on mobile CPU and GPU can be up to 48.55% and 47.65%, respectively. This saving is mostly from the reduced processing time and reveals that SMTM can achieve good performance on different devices.

### 8.6 Early exit performance
We also report the early exit performance of our SMTM across all the selected test benchmarks. Figure 11 shows the exit ratio of MobileNet V2 at different layers. During the CNN inference, we first record the exit position of each image and then summarize the exit ratio at each layer for the whole dataset. In Figure 11, the
absissa is the layer position, and the ordinate is the exit ratio. The final bar in the figure is the ratio of full inference. We perform semantics matching after each Inception module in MobileNet V2. Our experiments show that the exit ratio varies from layer to layer, which demonstrates that the semantic features of these middle layers have different characteristics. For the four CNN models on action recognition scenario, the exit ratio at all layers excepts the last one on the dataset ranges from 67.4% to 92.7%, and more than 50% of images on average can exit the inference in the first half of the network. For the extreme scenario image classification with rapid scene changes, there are also 63.8% images that can early exit the inference. The results indicate SMTM can make good use of the temporal redundancy in mobile videos to accelerate the CNN inference and well adapt to the various scenarios.

8.7 Choice of parameter
In our SMTM framework, the variable confidence threshold \( \tau \) for early exit can be used to make the trade-off between the latency reduction and accuracy loss. The threshold \( \tau \) is the key to decide whether the feature of the new input can be matched with the cached features of the selected objects in the fast memory. The results in Figure 12 show how \( \tau \) can affect the processing latency and accuracy.

8.8 The effect of adaptive memory
In this section, we evaluate the effect of the proposed two techniques in Section 6 to overcome the high dynamic scenarios in the mobile vision task.

First, we evaluate the effect of adaptive cache size in fast memory. We set a constant cache size (size=5) in the fast memory as 'SMTM (Constant)' and set this constant number as the initial cache size in 'SMTM (Adaptive)'. The other settings remain the same. The results in Table 1 shows that compared to 'SMTM (Constant)', 'SMTM (Adaptive)' increases the hit ratio by 21.6%, which brings 1.5× latency reduction.

|                | Hit ratio | Latency reduction |
|----------------|-----------|-------------------|
| SMTM (Constant)| 65.39%    | 25.21%            |
| SMTM (Adaptive)| 87.00%    | 38.46%            |

There is a trade-off between accuracy and latency when varying the fast memory size. Based on our experiments, we can indeed found an optimal fast memory size with the best trade-off in every small period of time. However, due to the high dynamicity in mobile data, the optimal size continues to change throughout the long video. In real-world applications, the optimal size is not known in advance. Therefore, we adopt probability estimation to predict a memory size in real-time in the paper. The predicted memory sizes are usually sub-optimal, but the overall performance exceeds the constant size by a large margin. In addition, the constant size...
in Table 1 is already the optimal size we found before, which can contribute to better performance on the entire test dataset.

Then, we evaluate the effect of adaptive semantic centers. The results in Figure 13 show that compared to the baseline (no-update), updating the semantic center adaptively can gradually improve the prediction accuracy and finally achieves 16.9% accuracy improvement on action recognition.

The above two experiments show the proposed two techniques are beneficial to the SMTM adapt to the scenario changes in mobile vision tasks.

9 DISCUSSION
This section highlights some of the limitations of SMTM and discusses possible future research directions.

**Generalized to diverse vision tasks.** Although SMTM currently focuses on recognition and classification tasks, we believe that the proposed memory design can be applied to many applications. First, recognition and classification are two general mobile vision scenarios, which include many applications involving deep learning in mobile/wearable devices. Second, for many multi-stage mobile vision tasks (such as object detection, etc), they usually also need to do recognition and classification. Thus, by leveraging the temporal redundancy in mobile videos, SMTM can also be potentially generalized to improve the processing in these tasks.

**Beyond vision tasks.** While SMTM currently focuses on continuous vision tasks, its key design of semantic memory can be potentially generalized beyond to other types of ML tasks such as natural language processing and speech recognition. This is because the temporal redundancy universally exists in those tasks, e.g., hot-spot keywords in the input method and voice assistants.

10 RELATED WORK

**CNN Cache.** A few recent works [9, 22, 51] also exploit the temporal redundancy to accelerate continuous visual tasks. They match the similar blocks (or pixels) of images or feature maps and reuse the intermediate partial results to skip computations. Those methods need to cache high-dimensional video frames and feature maps of CNN and then perform an expensive lookup on them. SMTM fundamentally differs from them in (1) SMTM encodes feature maps into low-dimensional semantic vectors of each object and executes distance measurement between these feature vectors. (2) SMTM directly exits the inference when an object is matched. FoggyCache [15] focuses on cross-device cache reuse, which is orthogonal to SMTM. EVA² [6] proposes a cache extension to CNN accelerator, which is not compatible with mobile scenarios. By contrast, SMTM is designed and implemented to run on general-purpose processors which are widely available on commodity mobile/wearable devices.

**Multi-branch neural architectures.** Some efforts propose multi-branch architecture into neural network designs to accelerate the CNN inference [13, 26, 29, 44, 47]. For example, BranchyNet [43] trains a network that allows simple samples to exit inference at the early layers. Similarly, SkipNet [44] allows easy samples to skip some middle layers during the inference and BlockDrop [48] skips some middle blocks for easy samples. While SMTM also early exits at different layers, its rationale is fundamentally different from the above-mentioned multi-branch networks. Our SMTM leverages opportunity from temporal-dimension, where inferences exit when repeated objects are observed from recent memory. Our key insight is orthogonal to theirs. SPINN [26] proposes a distributed inference system that employs synergistic device-cloud computation and progressive inference to accelerate CNN inference. However, putting the input to the cloud will bring significant privacy concerns. By contrast, SMTM is proposed to benefit directly on-device CNN inference. Furthermore, these above methods usually need to retrain the CNN models, which is a quite time-consuming process. SMTM is compatible with commodity, pre-trained CNN models, requiring zero effort from developers.

**On-device CNN optimizations.** Besides the aforementioned methods, extensive efforts have been made to optimize the CNN inference so they can be affordable on mobile/wearable devices, such as weight quantization [8, 10, 23], pruning [16, 30, 32, 36], hardware-based acceleration [50, 58, 59], model compression for mobile devices [28, 34, 53, 55, 60], etc. To our knowledge, SMTM is the first system that accelerates continuous mobile vision by exploiting the priming effect mechanism with CNN inference and is orthogonal to existing model-level or hardware-level optimizations.

11 CONCLUSIONS
In this paper, we propose a novel memory mechanism, called semantic memory, to speed up on-device CNN inference. The design of our memory mechanism is based on the observation of high temporal redundancy of continuous visual input on mobile scenarios and how human brains perform fast recognition of repeatedly presented objects. Extensive experiments show the superior performance of our system against existing cache systems.

ACKNOWLEDGMENTS
Mengwei Xu was supported in part by the Fundamental Research Funds for the Central Universities and National Natural Science Foundation of China under grant numbers 62032003, 61922017, and 61921003.

REFERENCES

[1] 2020. FFmpeg: a video processing platform. https://www.ffmpeg.org/.
[2] 2020. ncnn: a high-performance neural network inference framework. https://github.com/Tencent/ncnn.
[3] 2020. Vulkan API. https://github.com/KhronosGroup/Vulkan-Docs.
[4] 2021. General Data Protection Regulation (GDPR). https://gdpr-info.eu/.
[5] John A Bargh and Tanya L Chartrand. 2000. Studying the mind in the middle: a practical guide to priming and automaticity research. Handbook of research methods in social psychology. Handbook of research methods in social and personality psychology (2000), 253–285.
[6] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. 2018. EVA²: Exploiting Temporal Redundancy in Live Computer Vision. In 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). IEEE, 533–546.
[7] Kaidi Cao, Colin Wei, Adriend Gaidon, Nikos Ardechiga, and Tengyu Ma. 2019. Learning Imbalanced Datasets with Label-Distribution-Aware Margin Loss. In Advances in Neural Information Processing Systems (NeurIPS).
[8] Shijie Cao, Lingxiao Ma, Wencong Xiao, Chen Zhang, Yuxin Liu, Lintao Zhang, Lanshun Nie, and Zhi Yang. 2019. SeerNet: Predicting Convolutional Neural Network Feature-Map Sparsity Through Low-Bit Quantization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 11216–11225.
[9] Lukas Cavigelli, Philippe Degen, and Luca Benini. 2017. Chiinfer: Change-based inference for convolutional neural networks on video data. In Proceedings of the 11th International Conference on Distributed Smart Cameras (ICDSC). 1–8.
[58] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA). 161–170.

[59] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2018. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38, 11 (2018), 2072–2085.

[60] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroidLite: a release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys). 463–475.

[61] Xiangxin Zhu, Dragomir Anguelov, and Deva Ramanan. 2014. Capturing long-tail distributions of object subcategories. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 915–922.