Palleon: A Runtime System for Efficient Video Processing toward Dynamic Class Skew

Boyuan Feng, Yuke Wang, Gushu Li, Yuan Xie, and Yufei Ding,
University of California, Santa Barbara

https://www.usenix.org/conference/atc21/presentation/feng-boyuan

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the 2021 USENIX Annual Technical Conference is sponsored by USENIX.

Palleon: A Runtime System for Efficient Video Processing toward Dynamic Class Skew

Boyuan Feng, Yuke Wang, Gushu Li, Yuan Xie, and Yufei Ding
University of California, Santa Barbara
{boyuan,yuke_wang,gushuli,yuanxie,yufeiding}@ucsb.edu

Abstract

On par with the human classification accuracy, convolutional neural networks (CNNs) have fueled the deployment of many video processing systems on cloud-backed mobile platforms (*e.g.*, cell phones and robotics). Nevertheless, these video processing systems often face a tension between intensive energy consumption from CNNs and limited resources on mobile platforms. To address this tension, we propose to accelerate video processing with a widely-available, but not yet well-explored runtime input-level information, namely *class skew*. Through such runtime-profiled information, it strives to automatically optimize CNNs toward the time-varying video stream. Specifically, we build Palleon, a runtime system that dynamically adapts and selects a CNN model with the least energy consumption based on the automatically detected class skews, while still achieving the desired accuracy. Extensive evaluations on state-of-the-art CNNs and real-world videos demonstrate that Palleon enables efficient video processing with up to 6.7× energy saving and 7.9× latency reduction.

1 Introduction

Convolutional neural networks (CNNs) based video processing plays an important role in many emerging applications [3,4,7,9,10,16,31,35] deployed on cloud-backed mobile platforms. Among them, cognitive assistants and robotic visions are two representative categories. Smart glasses [7, 16, 31], for example, continuously recognize the surrounding environment with CNNs and help the blind person with ordinary tasks (*e.g.,* reading a handwritten note, navigating the grocery store, and even running the Boston Marathon). Robotic visions could automatically search specific animals and document the secret lives of them in the wild [9], as well as detect landmines in various environments [35].

While these applications enjoy both the mobility of the wide deployment in real world and the high accuracy of CNNs, they also face the tension between the limited resource budget on mobile platforms and the high energy consumption and latency of CNNs. A popular CNN, VggNet [67], can easily consume 3.6W and introduce 1.4-second latency, which makes a large smartphone battery (*e.g.*, 2.7-Ah battery in iPhone X [36]) out of power within 2 hours on continuous image classification. To improve the execution efficiency, many techniques have been proposed such as pruning [28, 46, 48] and quantization [56,76,80] to reduce the size of CNN models. However, these existing works fail to exploit the special characteristics of video streams; furthermore, the compromised model accuracy also limits the overall pruning or quantization ratio.

Complementing existing model compression techniques, a strong *temporal locality* in video streams is investigated here to enable efficient video processing on mobile platforms. Considering a video stream collected from a continuous camera feed, it is common that only a small number of classes keep appearing in a large number of consecutive frames. For example, in a film scenario, only a small number of people would come to the master shots frequently, generally lasting for a few minutes, and another group of people will not appear until the scenario has changed. A study on Youtube videos of day-to-day life [65] also shows that more than 90% frames are comprised of less than 10 classes.

We first turn such an abstract concept, *temporal locality*, into something concrete and measurable, *class skew*. Specifically, class skew is formally defined as an unbalanced class distribution that flexibly and effectively extracts the scenario information of both *class cardinality* and *visual separability*. Class cardinality here captures the number of classes in a class skew. By exploiting this class cardinality, we can tailor a general CNN, which is usually trained to recognize thousands of classes, into a specialized model, which only needs to recognize a small number of classes in the current class skew. Meanwhile, we notice that class skews show diverse visual separability under the same class cardinality. For example, a class skew with two classes (*e.g.,* houses and dogs) is easier to recognize compared to that with two more subtle classes (*e.g.*, Husky and Alaskan). By exploiting visual separability, we can use a more compact model for computation and energy saving without loss of accuracy compared with a full model.

We then identify several key challenges that hinder the

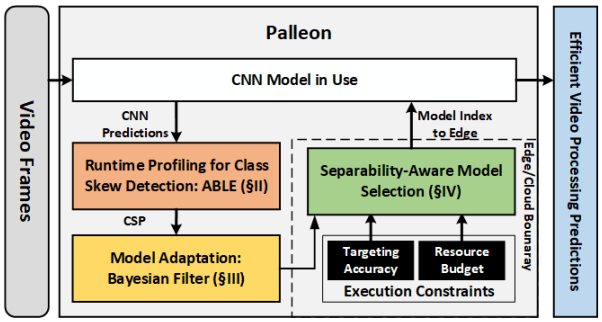


Figure 1: Overview of the Palleon Runtime System.

successful utilization of class skews. First, new class skews may appear and disappear suddenly as time goes, namely *class skew switches*, making it hard to precisely capture the class skew and respond fast to class skew switches. Second, a class skew may last for minutes or even hours between two class skew switches, but this lasting time varies across different videos and even scenarios, thus cannot be decided offline. Third, with the detected class skew, it is still hard to adapt deep models at runtime, since existing model adaptation techniques of retraining fully connected layers are computation-intensive and not affordable on mobile platforms. Forth, a single model adapted toward various class skews may show a significant difference in accuracy due to the diverse visual separability. This difference in accuracy either allows more lightweight CNNs for more energy saving or requires more computation-intensive CNNs to achieve a satisfactory accuracy. Finally, model selection adaptive to class skews may introduce high overhead, making it infeasible to execute on mobile platforms.

To address these challenges, we build a runtime system, Palleon, which could not only detect class skews during runtime, but also dynamically adapt and select a CNN model with the least energy consumption accordingly. In contrast, some existing works [32, 42, 43], which share a similar high-level motivation with us, only target some specific application scenarios that are already known offline—If you want to, you could think of these as "static" class skews without dynamic switches. A recent work, FAST [65], has made some progress along this research direction. FAST assumes that the exact set of class skews (and their duration time) in a video stream are foreknown, and FAST trains a set of compact models for each known class skew offline. During runtime, FAST only needs to detect these foreknown class skews with a simple window-based detector and directly apply those pre-trained models accordingly. To this end, Palleon adopts a pure runtime approach and targets a more realistic setting that the class skews in the video stream are not foreknown.

As illustrated in Figure 1, the Palleon runtime system continuously takes video frames and efficiently generates video processing predictions with three novel components. First, we propose an agile class skew detector, **ABLE** (Section 2), to abstract class skews from video streams. ABLE comes with the *static class-skew profiling* and the *dynamic class-skew switch detection*. The former automatically detects the class skew and generates a precise *Class Skew Profile* (CSP) when a class skew is detected. The latter continuously catches class skew switches during runtime without the offline information about the class skew lasting time.

Second, we propose **Bayesian Filter** (Section 3) to adapt CNNs toward the detected class skew during runtime. While Bayesian Filter does not directly lead to energy efficiency, it improves the accuracy of compact models with low resource consumption and allows the compact models to replace the complex model. Bayesian Filter is a lightweight module comprised of a *Rescaling* mode that adapts CNNs towards detected CSP without online finetuning and a *Direct Pass* mode that allows the adapted CNNs to still recognize classes out of the current CSP. This lightweight module resolves the complex trade-off between accuracy improvement and adaptation overhead in exploiting class skews.

Third, we design a cloud-backed model selection scheme, namely **Separability-Aware Model Selection** (Section 4), to further squeeze system energy consumption. This scheme exploits visual separability with an *efficient online model selection* that identifies the CNN with the least resource consumption while achieving satisfactory accuracy on the detected class skew. Meanwhile, this scheme contains an *edge-cloud duplicated model bank* to mitigate the model selection overhead on mobile platforms and deliberately schedule the runtime workload between the edge and the cloud.

In summary, we build Palleon, a runtime system that automatically detects input-level information with ABLE and dynamically adatps the given CNNs online with Bayesian Filter. Palleon also controls a set of tuning knobs for balancing the accuracy and the resource efficiency with separability-aware model selection. We build Palleon upon TensorFlow [1] and evaluate it on a cloud-backed mobile platform (with NVIDIA Jetson Nano [40] as the edge device and Dell Workstation T7910 [17] as the cloud server). We evaluate Palleon on various CNN models and different datasets. In particular, for CNN models, we use a variety of the state-of-the-art CNNs from two major domains – object classification (MobileNet [37], VGGNet [67], ResNet [30], and DenseNet [33]) and face recognition (VGGFace [57]). For datasets, we take both synthesized videos and several real-world movies. Extensive experiments confirm the effectiveness of Palleon and show that it could achieve up to 6.7× energy saving and 7.9× latency reduction while achieving an equivalent or better accuracy.

2 ABLE for Class Skew Detection

We build a class-skew detector, namely *ABLE*, to detect class skews during runtime and enable class-skew based optimizations. Our goal is two-fold: 1) giving a precise class-skew profile (CSP) in static regions between adjacent class-skew switches, and 2) detecting when the class-skew switches occur. To this end, we break down our class-skew detection into two sub-tasks: **Static Class-Skew Profiling** and **Dynamic**

Class-Skew Switch Detection.

2.1 Static Class-Skew Profiling

A static CSP generates the distribution of classes in a *static region* where no class skew switch happens. Palleon approximates the CSP in each static region with an empirical distribution [64], which enjoys theoretical properties of converging fast to the ground truth CSP. As illustrated in Figure 2a, given a time window with $r_t = 10$ frames, we collect the predicted labels for each frame and count the frequency of each class. For example, E appears for 4 times out of 10 frames in total, leading to 0.4 for class E in the estimated CSP.

Formally, at time t, in a given frame window with r_t frames (ranging from the $t - r_t + 1^{th}$ to the t^{th} frame), the probability of class j in the CSP is computed as

$$p(j|r_t, x_{1:t}) = \frac{1}{r_t} \sum_{i=t-r_t+1}^{t} \mathbb{1}_{x_i=j} \tag{1}$$

where x_i is the *predicted label* for the i^{th} frame, $\mathbb{1}_{x_i=j}$ is an indicator function [62] on whether x_i equals j, and $x_{1:t}$ denotes all t predicted labels in the history. The CSP for the given frame window (with r_t frames ending with the t^{th} frame) is computed as a probability vector of all classes:

$$CSP_{t,r_t} = \{p(1|r_t, x_{1:t}), p(2|r_t, x_{1:t}), ..., p(d|r_t, x_{1:t})\} \tag{2}$$

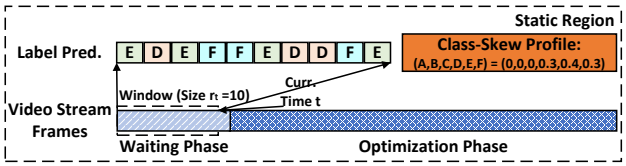
where d is the total number of classes.

Early Optimization by Adaptive Waiting Scheme. Palleon splits each static region as two phases (Figure 2a): a *waiting phase* to collect a precise CSP based on the full model and an *optimization phase* to apply class-skew based optimizations. In the optimization phase, we use a compact model adapted toward CSPs, which saves energy and reduces latency while achieving an equivalent accuracy to the full model. By allocating smaller number of frames in the waiting phase, Palleon can start the optimization phase early and squeeze more optimization opportunities for more frames, leading to better system performance. However, an imprecise CSP may be generated when allocating too few frames to the waiting phase. Hence, we select the frame number carefully to improve system performance and retain precise CSPs.

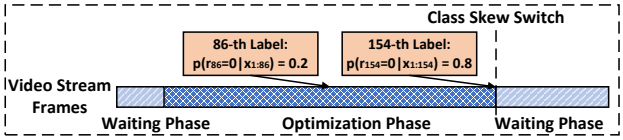
We develop an *adaptive waiting scheme* to determine whether Palleon has collected a precise CSP and the waiting phase can be terminated. Suppose the waiting phase has already lasted r_t frames and the current class-skew profile is CSP_{t,r_t}, this scheme computes a minimal frame number F_{min} based on CSP_{t,r_t}. If $F_{min} < r_t$, it terminates the waiting phase. Otherwise, it continues and repeatedly applies such check. In principle, F_{min} guarantees a negligible profiling error ε between the true probability p_j and the profiled probability $\hat{p}_j$

$$\max_{1 \le j \le d} |\hat{p}_j - p_j| / p_j \le \varepsilon \tag{3}$$

We next discuss how F_{min} is computed and why it guarantees a negligible profiling error. In addition, we will also



(a) Static Class-Skew Profiling.



(b) Dynamic Class-Skew Switch Detection.

Figure 2: Illustration of ABLE for Class Skew Detection.

propose some practical designs for efficiently computing F_{min}. We start with a theorem, which gives the minimum number of frames to profile a particular class in the class skew.

Theorem 1. (Asymptotic Error Bound). With $n = Z_c/(\varepsilon\sqrt{\hat{p}_j})$ samples (frames), the probability of achieving a negligible error $P(|\hat{p}_j - p_j|/p_j < \varepsilon) > 1 - c$ for class j holds asymptotically, where Z_c is a Gaussian Distribution Z-score with confidence level $1 - c$ and ε is a tolerable error bound.

Proof. Due to the property of multinomial distribution, estimator $\hat{p}_j$ computed with Equation 1 is a maximum likelihood estimator (MLE). Based on the asymptotic normality of MLE, we have $\sqrt{n}(\hat{p}_j - p_j) \to N(0, FI^{-1})$, where FI is the Fisher Information matrix $FI_{wh} = E_X[-\frac{\partial^2 ln f_p(X_t)}{\partial p_w \partial p_h}]$. Clearly $FI_{wh} = n/p_w$ if $w = h$; 0, otherwise. Based on the marginalization property of multivariate normality, we can see that $(\hat{p}_j - p_j) \to N(0, \frac{p_j}{n^2})$. Following this asymptotic distribution, we can derive the required sample number $n = Z_c/(\varepsilon\sqrt{\hat{p}_j})$.

Considering that CSP is stable only if estimators for most classes j are stable, we set the minimum number of frames as

$$F_{min} = \max_{\hat{p}_j > \xi} Z_c/(\varepsilon\sqrt{\hat{p}(j|r_t, x_{1:t})}) \tag{4}$$

Here we only consider classes showing *significant existence* ($\hat{p}_j > \xi$, where $\xi = 1/(2 * d)$ is a probability threshold). The intuition is that *CNNs may make wrong predictions randomly spanning in various classes with significantly low probabilities, leading to an unnecessarily long waiting time.* This strategy can mitigate the effect of prediction errors and focus on the effect of correct predictions.

For an arbitrary class number d, computing F_{min} has a low time complexity of $O(d)$, where the main computation resides in iterating through all classes j (Equation 4) and estimating the probability $\hat{p}(j|r_t, x_{1:t})$. This estimation can be conducted efficiently in constant time, based on a *computation reuse* technique detailed in the following section.

2.2 Dynamic Class-Skew Switch Detection

Dynamic class-skew switch detection identifies class skew switches and provides static regions for the static class skew profiling, as shown in Figure 2b. Specifically, dynamic class-

skew switch detection identifies the timestamp t when the previous class skew ends and a new class skew appears. There are two standard techniques to detect class skew switches: Window-based approach [5,65] and Bayesian-based approach [2,19,39,49,63,78]. The former splits video streams into a sequence of windows with a fixed window size k and periodically detects the class skew switch at the boundary. The latter detects class skew switch at each timestamp t by tracking all historical windows, where the $k^{th} \in \{1,2,...,t\}$ historical window contains the $t-k+1^{th}$ frame to the t^{th} frame. However, the former only detects the class skew switch when a time window has finished, leading to a detection delay up to the fixed window size. While the latter reacts fast to class skew switches by tracking all historical windows, it introduces high overhead with a quadratic time complexity $O((d+t)*t)$, in the number of frames t and the number of classes d. The latter shows more than 1500-millisecond latency per frame after processing a 3-minute video clip. By contrast, Palleon checks class skew switches at each label prediction x_t (Figure 2b) while introducing low computation complexity of $O(d*k)$, where k is the number of sampled windows ($k \ll t$).

At a high level, we sample a subset of window sizes $r_t \in \{w_1, w_2, ..., w_k\}$ and flag a class skew switch when the probability $p(r_t = 0|x_{1:t})$ is higher than the probability of the other r_t. We estimate the probability $p(r_t|x_{1:t})$ of each window size r_t when a new predicted label x_t comes:

$$p(r_t|x_{1:t}) = p(r_t, x_{1:t}) / \sum_{r_t=0}^{t} p(r_t, x_{1:t}), \tag{5}$$

where $p(r_t, x_{1:t})$ is the joint possibility of the lasting time r_t and the predicted labels $x_{1:t}$:

$$\begin{aligned} p(r_t, x_{1:t}) = \sum_{i=1}^{k} p(r_t|r_{t-1} = w_i) \cdot \\ p(x_t|r_{t-1} = w_i, x_{1:t-1}) \cdot p(r_{t-1} = w_i, x_{1:t-1}) \end{aligned} \tag{6}$$

Here, $p(r_t|r_{t-1} = w_i)$ is a survival function [44] of the probability that a class skew of length r_{t-1} is still alive at r_t, and $p(x_t|r_{t-1} = w_i, x_{1:t-1})$ is the probability that the predicted label x_t comes from the same distribution as last r_{t-1} labels, computed by Equation 1.

Overhead Reduction by Window Sampling. Window sampling selects k windows (*i.e.*, $w_1, w_2, ..., w_k$) that minimize the mean absolute error between tracking the selected k windows and all t windows:

$$\min_{w_1, w_2, ..., w_k} \sum_{r_t=1}^{t} |p(r_t, x_{1:t}) - p_{f(r_t)}|, \tag{7}$$

where $f(r_t)$ maps time window r_t to one of the sampled time window $\{w_1, w_2, ..., w_k\}$. A small number of k windows can approximate all t windows since a window size with low possibility $p(r_{t-1}, x_{1:t-1})$ tends to still have low possibility $p(r_t = r_{t-1} + 1, x_{1:t})$ when new data comes:

$$\begin{aligned} p(r_t, x_{1:t}) &= p(r_t|r_{t-1}) p(x_t|r_{t-1}, x_{1:t-1}) p(r_{t-1}, x_{1:t-1}) \\ &\leq p(r_{t-1}, x_{1:t-1}) \end{aligned} \tag{8}$$

A straightforward approach is to only track the k most possible windows. However, this approach may not work well when a large number (> k) of windows have equally high probability $p(r_t, x_{1:t})$. To this end, we group all t windows into k clusters and select a representative window out of each cluster. To cluster windows, we first sort the possibility $p(r_t, x_{1:t})$ for all time windows r_t and split into clusters at the top $k-1$ gaps in the sorted sequence. Then, we select the windows with the median probability to represent this cluster and give this window a weight based on the number of windows in the cluster. Intuitively, we exclude the top $k-1$ gaps by splitting clusters at these gaps to minimize the mean absolute error.

Fast Update by Computation Reuse. Another optimization opportunity is the computation reuse in estimating the conditional distribution $p(x_t|r_{t-1}, x_{1:t-1})$. Since the estimation for each window r_{t-1} at most traverses all data points and all classes, the time complexity of the estimation is linear to the number of data points and classes $O(d+t)$. Since adjacent windows differ only by one input, we can reuse the class frequency in adjacent windows, reducing the time complexity for each window to be $O(d)$. Specifically, the class frequency counts in adjacent windows $r_{t-1} = i$ and $r_{t-1} = i+1$ differ only by one in a single class, determined by the label x_{t-i}. Thus, with the class frequency count $[C_1, C_2, ..., C_d]$ in window $r_{t-1} = i$, we can update the frequency count in window size $r_{t-1} = i+1$ by $C'_j = C_j + 1$, if $j = x_{t-i}$; $= C_j$, o.w.

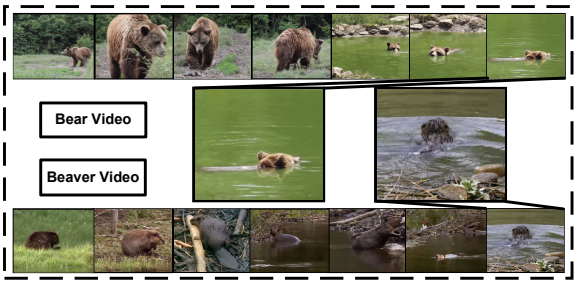
3 Bayesian Filter for Model Adaptation

In this section, we develop a highly flexible module, namely *Bayesian Filter*, to enable class-skew based optimizations. This module consumes a Class Skew Profile (CSP) detected by ABLE and has two functionalities: 1) adapting CNNs toward the detected class skew with low computation overhead and low latency during runtime; 2) allowing the adapted CNNs to recognize classes out of the current CSP for enabling the detection of class skew switches. To this end, we design a **Rescaling** mode (Section 3.1) for scenario reference (*i.e.*, model adaptation) and a **Direct Pass** mode (Section 3.2) for retaining confident predictions.

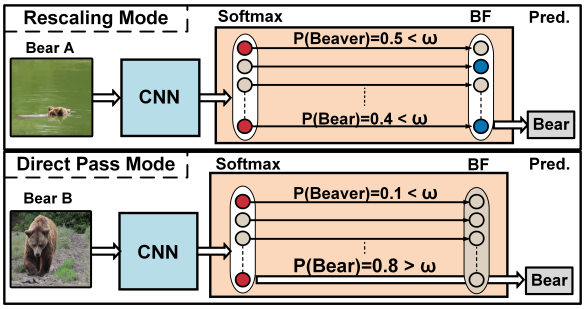
Figure 3a exhibits two videos with extreme class skews, which are intentionally made simple for understanding. In most video frames, the objects are easy to recognize, during which Palleon extracts a precise CSP based on classes recently detected with high confidence. This CSP helps for frames in which objects are hard to recognize (*e.g.*, the animals jump into the water and turn around).

3.1 Rescaling

Rescaling mode is a lightweight module for model adaptation with low computation overhead and low latency. This design avoids the heavy computation overhead and latency in existing work [27, 32, 43, 65], which retrains fully connected layers in CNNs during online and may introduce a long latency (up to 14 seconds) [65]. When considering the energy efficiency,



(a) Two videos with extreme class skews



(b) Intuition behind Bayesian Filter.

Figure 3: Overview of Bayesian Filter.

this retraining procedure either introduces heavy computation overhead when conducted on the edge, or heavy network communication overhead when retraining is conducted on the cloud and updated weights are transferred to the edge. By contrast, Rescaling mode adapts CNNs by appending an extra layer after the fully connected layer. We stress that Rescaling mode requires neither weight update nor model retraining.

Rescaling mode adapts CNNs toward the detected class skew by initializing the extra layer with the CSP generated by ABLE. Since CSP contains the probability of all d classes, the extra layer is designed to have the same number of d nodes, whose weights are initialized by each probability in CSP. In this way, the magnitude of node weights indicates the frequency of the corresponding class in the CSP. When a frame comes, the CNN will generate a probability for each class in the softmax layer and these probabilities will be adjusted according to the magnitude of corresponding node weights. Specifically, the extra layer rescales the probability of softmax-layer predictions (red nodes) toward the current CSP (blue nodes) when the highest softmax-layer probability does not pass a pre-defined threshold ω (*i.e.*, not confident enough). Following the spirit of Bayesian statistics [64], Rescaling mode updates the probability for each class by considering both the prior and the posterior information. We tried several different designs, and it turns out that a simple rescaling scheme based on Bayes theorem would already work effectively, as shown in Formula 9.

$$P(i|X) = \frac{P(i) \cdot P(X|i)}{P(X)}, \quad i \in \{1, 2, ..., d\} \tag{9}$$

where d is the total number of classes.

Formula 9 computes the posterior probability of class i for a given image X. The prior probability $P(i)$ is the profiled probability of class i in the current CSP. And the likelihood $P(X|i)$ describes the possibility that an image X comes from class i, according to the softmax-layer probability of class i. $P(X)$ stands for the marginal likelihood for observing the image X, which is same for all classes and does not change the rescaling results. Thus, we can avoid computing $P(X)$ and, instead, use a handy rescaling mechanism as $P(i|X) \propto P(i) \cdot P(X|i)$. To the best of our knowledge, we are the first to design Bayesian rescaling on CNNs for runtime model adaptation toward class skews.

3.2 Direct Pass

Direct Pass mode selects the original prediction without rescaling when the predicted probability is higher than a pre-selected threshold ω (Direct Pass mode in Figure 3b). This design allows detecting class skew switches by identifying classes out of the current CSP. This design is inspired by observations [25, 54] that *neural networks usually achieve higher accuracy when they predict a high probability.*

Formally, Bayesian Filter with both Rescaling mode and Direct Pass mode can be written as

$$P(i|X) \propto \begin{cases} P(i) \cdot P(X|i) & \text{if } P(X|i) < \omega \\ P(X|i) & \text{if } P(X|i) \geq \omega \end{cases} \tag{10}$$

where $P(i)$ is the prior probability of observing class i provided by class skew, $P(X|i)$ is the predicted probability from CNNs. We utilize a hyper-parameter ω as the confidence threshold deciding whether Bayesian Filter should enter the direct pass mode. When a model makes a prediction with high probability ($> \omega$), we believe its prediction is correct and Bayesian Filter will not interfere with the decision. Note that two models with different accuracy—for example, a model A with 70% accuracy and a model B with 95% accuracy—could predict with similar high probability when they are making the correct predictions. Their accuracy difference comes from those frames where the poorer model makes a mistake while the stronger model is still correct, not from those frames where both models are correct. Thus, we select the same threshold ω across different CNNs. In particular, we experiment with diverse ω on an extensive collection of the state-of-the-art CNNs and find that a threshold ω between 75% and 95% exhibits similar performance. By default, we use 90% as the threshold ω in following sections.

4 Separability-Aware Model Selection

We propose Separability-Aware Model Selection to enable class-skew based optimizations by exploiting the visual separability. The key observation is that, the same model under different class skew profiles (CSP), even with the same number of classes, may have significantly different accuracy. Figure 4 illustrates two CSPs with different visual separability (*i.e.*,

Figure 4: Class Skews with Different Visual Separability.

one is easy to classify and the other one is hard). To exploit visual separability, Palleon maintains a set of models with different accuracy and energy consumption, and automatically switches to compact models for saving energy when the detected CSP is easy to classify. We are inspired by the fact that people will relax and spend less energy when objects have significantly different appearance, in contrast to distinguishing similar objects (*e.g.*, cat breeds). To this end, we propose an **Efficient Online Model Selection** (Section 4.1) to automatically select models with low resource consumption, and an **Edge-Cloud Duplicated Model Bank** (Section 4.2) to reduce model selection overhead and network overhead.

4.1 Efficient Online Model Selection

We conduct online model selection on the cloud when we detect class skew switches. There are two baseline strategies. One approach records an average accuracy for each model on all classes, and another approach records the accuracy of one model over all possible CSPs (*i.e.*, multiple accuracy for one model). Both approaches are unsatisfactory. On the former approach [27], the selected model may fail to satisfy the accuracy requirement during runtime since the same model may produce significantly different accuracy on different CSPs. On the latter approach [65], a prohibitive offline profiling overhead and online memory overhead may be introduced due to the huge number of CSPs.

By contrast, we propose a hybrid approach that selects models on the cloud for only class skews detected during runtime. During offline preparation, we profile a single accuracy for each model over all classes and store the model in the order of this accuracy. During online model selection, we use binary search to profile the CNN accuracy on the detected CSP. This binary search leads to logarithm time complexity, compared to the linear time complexity of enumerating all models. Behind the binary search, our key observation is that, while the model accuracy on each CSP may change dramatically, the relative accuracy order of models on all classes stays the same over various class skews. In particular, if one model performs better than another model on all classes, the former one generally performs still better than the latter model on various CSPs. Similar observations have also been made in computer vision area [30, 37] that larger models (e.g., ResNet-50) usually give higher accuracy than smaller ones (e.g., ResNet-18) on the same task. Figure 5 illustrates the online model selection. Suppose we have 5 models with decreasing energy consumption and recognition accuracy, and target 90% accuracy. For each CSP, we conduct binary search to find the most compact model with satisfactory accuracy (> 90%). In this case, we will select B for CSP I but A for CSP II.

	A	B	C	D	E
CSP I	97	91	87	84	81
CSP II	90	88	82	79	77

Figure 5: Example of Online Model Selection (Unit:%). Dashed boxes refer to un-profiled models due to binary search.

Cache Service to Avoid Redundant Model Selection. Palleon records the model selection results along with the CSP and skips model selection for a CSP that have appeared previously. In particular, Palleon maintains a cache service between the CSP and the selected model. When a new CSP comes, Palleon will first retrieve the CSP in the cache. On a cache hit, Palleon immediately returns the selected model. On a cache miss, Palleon conducts online profiling and records the selected model for reuse. In our evaluation, a high cache hit rate is achieved quickly after less than 5 model selections, since the same CSP appears frequently in real videos.

4.2 Edge-Cloud Duplicated Model Bank

Palleon's goal of saving energy and improving accuracy is affected by the quality of its model bank. We design a duplicated model bank to store only Pareto-Optimal models and duplicate these models on both the edge and the cloud for reducing network overhead. For each candidate model, Palleon stores the computation graph, the pre-trained weights, and the metadata including energy consumption and latency.

Model Bank Generation with Offline Profiling. For each energy budget, we conduct offline profiling to identify candidate models with the highest accuracy. This offline profiling selects only models on the Pareto-optimal curve to reduce online search space and runtime overhead. Specifically, we first generate a large number of candidate models by applying compression techniques on CNNs. Then, we conduct offline profiling to select models on the Pareto-optimal curve [55], defined as the models that we cannot further reduce energy consumption without worsening the accuracy.

This candidate model generation is *conducted once on all classes*, instead of repeating on different CSPs, since good models on all classes tend to consistently produce good performance over various CSPs. The insight is that *unsalient positions remain similar for all CSPs*. For example, when we repeat a compression technique, Perforation [20], for several CSPs on Dense-40 [33], the positions in later blocks will be deleted first while the positions in the leading block remain unchanged until all later blocks have been pruned. More generally, even though the best model (*i.e.*, the one with the highest accuracy under a given reduction in energy consumption) might change between different CSPs, the set of *top-k best* models tends to remain stable over all CSPs. Thus, we can avoid repeating the selection on all CSPs and, under any

Table 1: Profiling on selected compact models. Latency and energy are measured on Jetson Nano.

Compressed From	Layer Remove Ratio (%)	Filter Remove Ratio (%)	Latency (ms)	Energy (J)
ResNet-50	20	10	65.6	0.63
DenseNet-40	30	10	48.4	0.46
MobileNet-128	30	30	34.5	0.35
MobileNet-128	30	40	20.6	0.18
VggNet-19	60	60	10.1	0.07

specific energy-saving requirement, use the best model for all classes to approximate the best model for a specific CSP.

Our full model is a DenseNet-40 with 40 layers. Starting from 4 base models (i.e., MobileNet-128, VGGNet-19, ResNet-50, DenseNet-40), we generate 25 compact models from each of these base models. In particular, we first remove {10%, 20%, 30%, 40%, 60%} of layers from the base model. For the remaining layers, we remove {10%, 20%, 30%, 40%, 60%} of filters. While more sophisticated compression techniques can be applied, we adopt this simple compression technique to validate the effectiveness of model selection. From these pruned models, we select N_{CW} (=5, by default) compact models and put them into our model bank for online use in our evaluation. We have experimented with several numbers and found that 5 compact models can provide a relatively diverse range of accuracy and resource consumption. We show the profiling data on raw latency and raw energy consumption in Table 1, measured on Jetson Nano [40]. For each frame, these models have inference latency from 10.1 ms to 65.6 ms and energy consumption from $0.07J$ to $0.63J$. Here, all pruned models are retrained over all classes during offline model bank generation.

Edge/Cloud Duplication to Reduce Network Overhead. We maintain a duplicated model bank on both the edge and the cloud to avoid weight transportation from the cloud to the edge. The duplicated model banks on the edge and the cloud contain the same deep models and pre-trained weights, while giving each model an index. During online model selection, the cloud will select a model from the duplicated model bank and only send the selected index to the edge. The edge uses the received index to identify the selected model. This design avoids the network overhead of frequently transporting model weights when class skew switches frequently.

4.3 System Overhead Analysis

Model Bank Memory Overhead. The model bank introduces negligible memory overhead compared to the simple setting with only large CNNs. In the simple setting, the memory consumption is

$$Mem_{Simple} = Mem_{LW} + Mem_{LF} \tag{11}$$

where Mem_{LW} and Mem_{LF} are the memory for storing weights and features of the large CNN, respectively. In our setting with model bank, the memory consumption is

$$Mem_{Bank} = Mem_{LW} + max(Mem_{LF}, Mem_{CF}) + N_{CW} \cdot Mem_{CW} \tag{12}$$

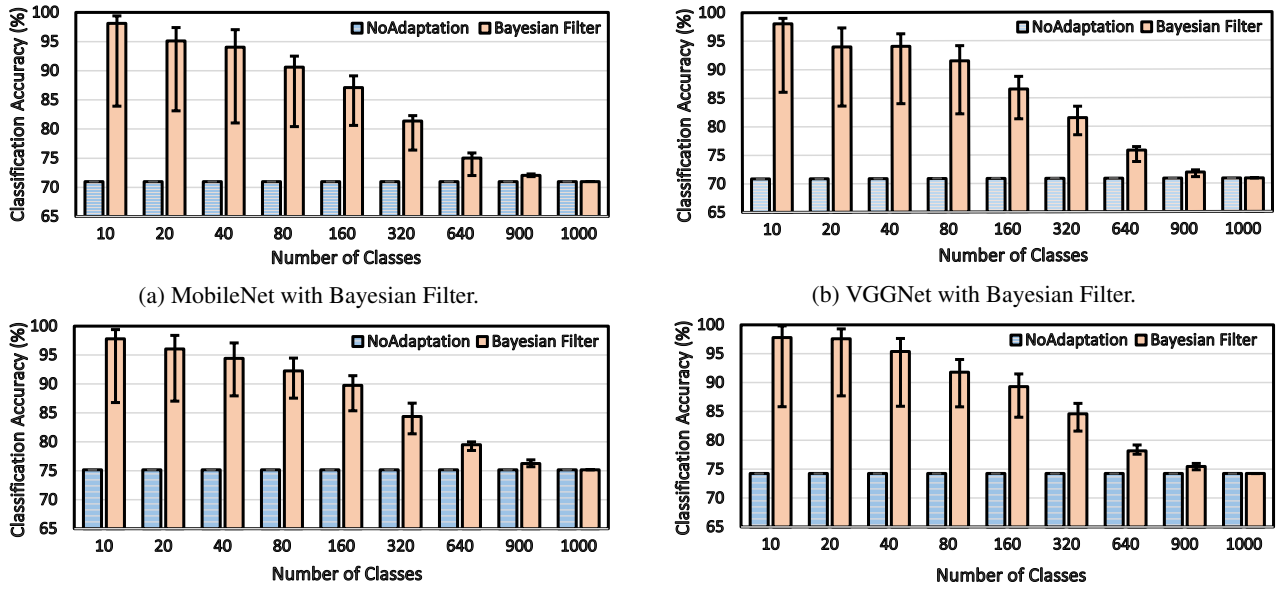
where N_{CW} is the number of compact models, Mem_{CW} and Mem_{CF} are the memory for storing weights and features of compact CNNs. We use $max(\cdot,\cdot)$ on CNN features since each input frame is processed by only one CNN. Comparing Equation 11 and 12, the model bank only introduces overhead of $N_{CW} \cdot Mem_{CW}$, which is less than 5MB and is negligible compared to the GB-level memory in modern edge devices (GB) (*e.g.*, 1GB in Raspberry Pi 3B+ [59] and 4GB in Jetson Nano [40]). In particular, we use a small N_{CW} (=5, by default), since Bayesian Filter can adapt compact models toward the detected CSPs during runtime with low overhead (Section 3). The Mem_{CW} is usually less than 1MB on compact models generated with compression techniques, especially when the base model also consumes negligible memory (*e.g.*, 0.5MB in SqueezeNet [34] and 2MB in MobileNet [37]).

Runtime Overhead. Palleon's runtime overhead comes from three sources. The first is the model selection overhead. While this overhead is relatively large, model selection is conducted on the cloud which is powerful and can evaluate several models concurrently. Also, we have sorted the models and proposed a binary search for accelerating the model selection. This procedure introduces negligible runtime overhead (<1%). The second is the data transfer overhead. Existing work usually transfers frames (around 100 KB per frame) to the cloud, which introduces heavy network overhead. Instead, we summarize the surrounding environment into the CSP (a short string within 1KB) and only need to transport the CSP from the edge to the cloud through a wireless network. This network overhead is low since we only transport CSP instead of data or CNN weights. Besides these two overhead from model selection, the third comes from class skew detection on the edge, which is negligible due to optimizations for low-overhead detection in Section 2.2.

5 Evaluation

To show the effectiveness of Palleon, we perform extensive experiments on both synthesized videos and real videos. We first evaluate Palleon on **synthesized videos** (Section 5.1) to study the performance in diverse settings, including varying class numbers, class types, and lasting time of each class skew. We then conduct **real video experiments** (Section 5.2) to further validate the performance of Palleon for detecting and exploiting class skews during runtime.

Experiment Platform. We have implemented Palleon in Tensorflow [1] for our CNNs. For the edge device, we use NVIDIA Jetson Nano [40] which is a popular mobile GPU platform with wide deployment in robotics [53], AI glasses [52], and doorbell cameras [51]. Jetson Nano runs Ubuntu 18.04 with built-in support for Tensorflow. For the cloud server, we use a Dell Workstation T7910 with an NVIDIA



(a) MobileNet with Bayesian Filter.

(b) VGGNet with Bayesian Filter.

(c) ResNet with Bayesian Filter.

(d) DenseNet with Bayesian Filter.

Figure 6: Accuracy on Fixed Class Skews. Error bars represent the accuracy range for each number of classes.

1080Ti GPU (with 11 GB dedicated memory and a nominal peak performance of 11.3 TFLOPS), a 6-core Intel Xeon CPU E5-2603 processor with 32 GB memory running Ubuntu 18.04. All energy measurements mentioned are directly measured unless otherwise specified, using an Extech EX330 Compact Digital Multimeter [50].

5.1 Synthesized Video Experiments

In this section, we extensively evaluate Palleon on synthesized videos in diverse settings. We generate synthesized videos based on ImageNet dataset [18] with diverse class skews. The ImageNet dataset consists of $1,200,000$ images categorized into $1,000$ classes. We generate class skews with varying numbers of classes and diverse lasting time. These synthesized class skews create challenging scenarios and showcase the robustness of Palleon in challenging settings. We train CNNs on ImageNet with all classes and conduct offline profiling to generate a single accuracy over all classes and collect their latency/energy consumption on mobile devices (e.g., Jetson Nano). This offline profiling is conducted only once. During online, we use Bayesian Filter for online model adaptation and do not use online finetuning (*i.e.*, retraining CNNs on the detected CSPs). On dynamic class skews, we also have online profiling about the model accuracy on the detected CSP, which is conducted on the cloud and introduces negligible overhead.

5.1.1 Bayesian Filter on Fixed Class Skews

Figure 6 shows the accuracy improvement from Bayesian Filter in an ideal case that the true class skew is known and fixed. Under this setting, there is no detection delay and Bayesian Filter can adapt CNNs toward the true class skew. To show the generality of Bayesian Filter, we use four state-of-the-art CNNs as base models (*i.e.*, MobileNet [37], VGGNet [67], ResNet [30], and DenseNet [33]). When synthesizing fixed class skews, for each $N \in \{10, 20, ..., 1000\}$ classes, we generate 100 CSPs. Each CSP contains 1000N images by randomly selecting N classes and 1000 images from each class following a uniform distribution. For each number of classes, we run the adapted model and present the average, minimum, and maximum accuracy. We note that we use Bayesian Filter to adapt the model and do not use online finetuning.

Bayesian Filter provides on average 25% accuracy improvement when there are 10 classes. This accuracy improvement shows the effectiveness of Bayesian Filter in adapting models. As the number of classes increases, this accuracy improvement diminishes gradually until all $1,000$ classes appear in the class skew (*i.e.*, no scenario information to exploit). The reason is that, as the number of classes increases, opportunities to rule out classes decreases. For example, Bayesian Filter rules out 990 classes when the CSP contains 10 (out of 1000) classes, but only rules out 100 classes when the CSP contains 900 (out of 1000) classes. Surprisingly, an accuracy improvement around 2% still exists when there are 900 classes in the class skew, considering that underlying models can only recognize $1,000$ classes. This result shows that Bayesian Filter can consistently improve accuracy on challenging class skews with a large number of classes.

A large accuracy difference exists for each model and each number of classes, demonstrating the existence of visual separability. In particular, an accuracy difference of 15% can be observed for MobileNet when there are 10 classes. This accuracy difference becomes less significant as the number of classes increases, since a smaller number of classes indicates larger variation in the constituent classes. Comparing across models (*e.g.*, ResNet v.s. MobileNet), we see that a model tends to perform better than another model on a specific class

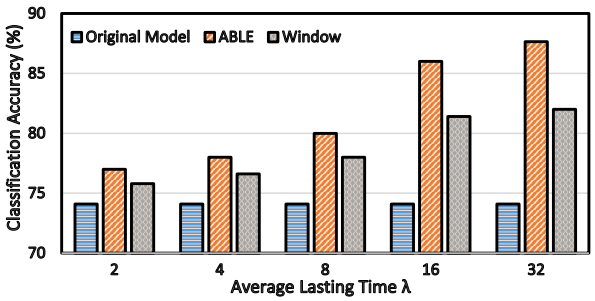


Figure 7: Accuracy with Various Detection Methods.

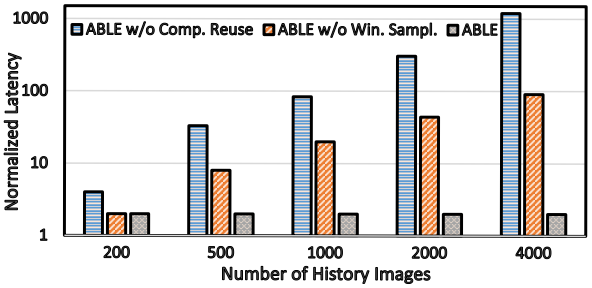


Figure 8: ABLE Latency relative to Window Detector.

skew, if the former model has higher accuracy on all classes than the latter model. This observation indicates that the relative order of model accuracy is invariant over various class skews and supports our binary profiling.

5.1.2 ABLE on Dynamic Class Skews

In this section, we show the accuracy improvement when the true class skew is unknown and may switch abruptly, namely *dynamic class skew*. Under this setting, the class skew detector decides the CSP quality and the detection delay, which has a significant impact on the classification accuracy of the adapted models. When synthesizing dynamic class skews, we randomly generate 30 CSPs. For each CSP, we first randomly select a small number (ranging from 10 to 20) of classes. Among all testing images from a given set of classes, a CSP uniformly samples $lastingTime = 60 * T$ images. T is a random variable following the Poisson(λ) distribution and $\lambda \in \{2,4,8,16,32\}$ controls the average number of images. We choose the Poisson distribution, as it outputs positive integers. We use DenseNet as the original model and elide the results on other CNNs due to the similar behavior.

Average Accuracy. Figure 7 shows the classification accuracy on dynamic class skews when combining Bayesian Filter with two class skew detectors (*i.e.*, Window and ABLE). In the Window detector, we sample a sequence of window sizes for each synthesized video and present the best accuracy for a strong baseline. Comparing across λ, we can see a clear trend that the classification accuracy increases as λ increases. In particular, "ABLE" can increase accuracy by 13.65% when the average lasting time λ reaches 32. This trend indicates that a class skew lasting longer provides more optimization opportunities to exploit. Comparing across detectors, we can see that ABLE achieves higher accuracy improvement around

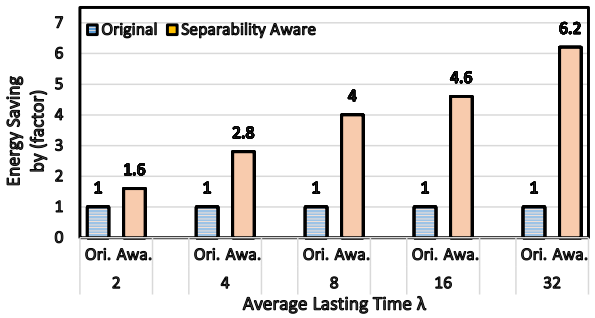


Figure 9: Energy Saving with Model Selection.

5% than the window-based detection. The reason is that the lasting time for a specific class skew varies even for a fixed average lasting time λ, such that a fixed window size can hardly hit the balance between CSP quality and detection delay.

ABLE Detection Latency. Figure 8 shows the ABLE detection latency reduction. This detection latency measures the computation overhead of incurring ABLE on an incoming CNN prediction. *Number of history images* is the total number of images in a synthesized video representing the video length, ranging from 200 to 4000 images. "ABLE" represents ABLE with both computation reuse and window sampling where $k = 30$ windows are sampled. "ABLE w/o Win. Sampl." disables window sampling and "ABLE w/o Comp. Reuse" further disables computation reuse. "ABLE w/o Comp. Reuse" shows a quadratic increase in the latency over time, which becomes costly when the number of inputs increases gradually. By adding computation reuse, "ABLE w/o Win. Sampl." decreases this quadratic time complexity to linear time complexity, leading to a much lower computation overhead. When adding window sampling, we can see that "ABLE" further reduces the linear time complexity to a constant complexity, which is similar to the Window detector.

5.1.3 Separability-Aware Model Selection

In this section, we show the energy-saving, runtime speedup, and the memory overhead from Separability-Aware Model Selection. To study the impact of lasting time, we adopt the same setting as dynamic class skew (Section 5.1.2). In each dynamic class skew, we randomly generate 30 class skews and report the average energy saving and runtime speedup.

Energy Saving. Figure 9 shows energy saving when targeting the same accuracy as the baseline model. Palleon can save energy consumption up to 6.2× while maintaining the accuracy. This benefit comes from automatically replacing the original large model with small models by separability-aware model selection. In addition, we can observe that the energy saving increases as lasting time increases, since a longer lasting time indicates less class skew switches and less system overhead for detecting and exploiting new class skews.

Runtime Speedup. Figure 10 shows the overall runtime speedup when targeting the same accuracy as the baseline model. Palleon can achieve up to 5.48× speedup, consider-

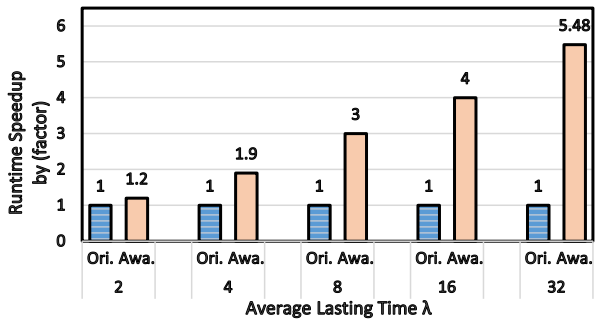


Figure 10: Runtime Speedup with Model Selection.

ing both the model execution speed and the system latency, including the model adaptation latency, the model selection latency on the cloud, and the network latency. Note that the model selection on cloud only introduces latency and has no impact on the energy consumption on the edge device. Similar to the observation in energy saving, we can observe an increase in runtime speedup as the lasting time increases, due to the reduced system overhead.

Memory and data transfer overhead. We observe negligible memory overhead in our current system with 5 compact models. In particular, these compact models usually consume less than 1MB memory and Jetson Nano has 4GB memory. On the data transfer overhead, we only transfer a short CSP (within 1KB) to the cloud when ABLE detects class skew switches. This is significantly smaller than alternative system designs that transfer frames (around 100 KB per frame) or CNN weights with the cloud.

5.2 Real Video Experiments

We evaluate Palleon on real videos to show the end-to-end accuracy improvement, runtime speedup, and energy saving, including the overhead from model adaptation and class skew detection. We compare Palleon with a state-of-the-art energy-efficient video processing system, FAST [65]. *FAST* approach studies the benefit of class skews in an ideal case, assuming that all class skews that may appear at runtime are known offline. In particular, FAST adapts a large number of compact CNNs towards each class skew during offline preparation and identifies these foreknown class skews with a window detector. As such, we denote it as *FAST (offline)*. For a fair comparison of our online framework, we further extend FAST (offline) to an online version, namely FAST (online). The only difference between these two versions is that FAST (online) does not have the pre-trained CNN models for different class skews. Instead, it adopts a standard retraining method [68], which retrains the last few CNN layers toward the online detected class skews, for online model adaption. To strike a good balance between accuracy and performance, we manually tune the number of layers for retraining and find that two is a good number and use it in our experiments. Different from FAST, we do not foreknow the class skews offline. We conduct online class skew detection with ABLE, online model adaptation with Bayesian Filter, and online model selection.

Table 2: Real videos for evaluating Palleon. "#Switch" indicates the number of class skews switches and "#Class" shows the average number of classes in each class skew.

Video Name	Len. (min)	#Switch	#Class
Friends	24	45	2.8
Good Will Hunting	14	4	3.5
The Departed	9	8	2.4
Ocean's Eleven / Twelve	6	25	2.0

Real Video Datasets. We evaluate Palleon and FAST on four real videos [65] depicted in Table 2. These videos come from several movies for face recognition and have diverse length ranging from 6 minutes to 24 minutes. "#Switch" indicates the number of class skew switches in each video and "#Class" represents the average number of classes (faces) in each class skew between adjacent class skew switches. For example, "Friends" is a 24-minute video with 45 class skew switches and each class skew contains 2.8 classes on average. While the total number of classes in these real videos is large (> 20), each class skew contains only a small portion of classes (2 to 3.5). This is a common case in films and Youtube videos, as we have discussed in introduction. The lasting time for each class skew varies on videos from 10 seconds to 4 minutes (about 1.3 minutes on average), computed by "Len.(min) / #Switch". This diversity makes it a challenging setting to detect and exploite class skews during runtime.

During the detection of faces, we follow the standard two-phase pipeline in object detection [23, 24, 60] and face recognition [6]. We first use a Viola Jones detector [70] to locate faces in video frames, which is agnostic to class skews. Then we crop faces and feed into CNNs for face recognition [57]. Note that this procedure can be easily applied to other object detection tasks by retraining the face recognition CNNs.

Base Model for Real Video Datasets. For a fair comparison with FAST, we choose the state-of-the-art deep model, VGGFace [57], as our full model for face recognition. We generate 5 compact models with diverse resource consumption and accuracy, following the model bank generation in Section 4.2. We use the same compact models for FAST. We train these models from scratch following the hyper-parameter setting in FAST, and achieve comparable accuracy as reported. This offline training is conducted once on LFW dataset. During online, we use Bayesian Filter for model adaptation and do not use online finetuning.

Accuracy Improvement. Figure 11 shows the overall classification accuracy on real videos of the most *compact VGGFace* model, FAST (offline), and *our online approach*. We skip the accuracy of FAST (online) since it consistently provides lower accuracy than the Fast (offline) by around 1%, since we retrain only the last two layers in Fast (online) to hit a good balance between accuracy and performance. Palleon provides 6.2% accuracy improvement on average, compared

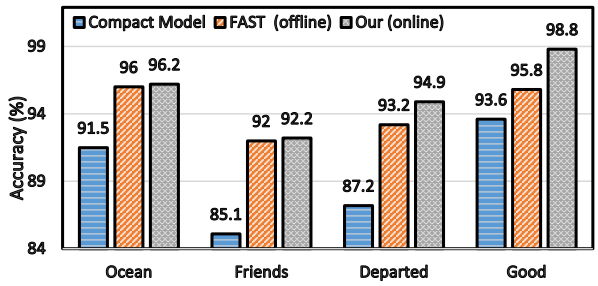


Figure 11: Accuracy Improvement on Various Videos.

to utilizing the compact model without adaptation. This accuracy improvement comes from Bayesian Filter that dynamically adapts models to class skews detected in real videos, containing only 2 to 4 people on average ('#Class' in Table 2). This reduced number of faces greatly eases the task compared to recognizing thousands of faces in un-adapted models.

Comparing to FAST (offline) relying on offline adapted models, Palleon provides 1.3% accuracy improvement due to the faster class skew switch detection in ABLE. When a class skew switches, Window detector in FAST (offline) leads to a detection delay up to 10 seconds, during which the accuracy suffers from a dramatic drop. Moreover, ABLE can effectively detect 98% class skew switches while Window detector can only detect 86% class skew switches, since Window detector fails to detect class skews that exist for only a few seconds.

Runtime Speedup. Figure 12 shows the end-to-end runtime speedup on real videos when targeting the same accuracy as the full model. Palleon achieves on average 5.43× speedup (up to 7.9× speedup on Good) compared to the full model. This speedup comes from automated model selection by replacing the full model with a compact model. When both assuming that the class skews are not foreknown and adapting models at runtime, Palleon achieves 26.9× speedup over the FAST (online) approach. Indeed, FAST (online) shows a 5× slow down due to heavy overhead from online model adaptation. This comparison demonstrates the efficiency of Palleon in online class skew detection and online model adaptation. For FAST (offline) with strong assumption that true class skews are foreknown and models are adapted during offline preparation, Palleon can still achieve a higher speedup, due to the early optimization strategy in ABLE. Comparing across videos, the speedup becomes more significant when class skews have longer lasting time (Good Will Hunting), showing the same pattern as evaluations on synthesized videos (Section 5.1.3). We note that Palleon takes 14.2 ms latency on average to process one frame, which is significantly faster than the real-time requirement of 30 ms per frame.

We also observe negligible overhead from model switches (<1%) due to several reasons. First, the number of model switches is much smaller than the number of class skew switches, since class skew switches can usually be handled by the Bayesian Filter without model switches. In particular, only 20% class skew switches lead to model switches. Second, the

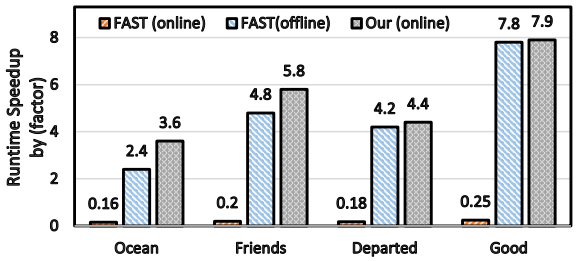


Figure 12: Runtime Speedup on Various Videos.

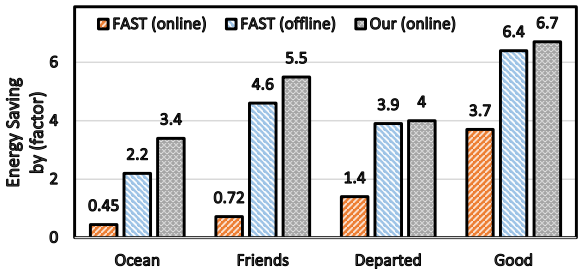


Figure 13: Energy Saving on Various Videos.

model selection is conducted on the cloud and we cache all models in the memory to avoid the repeatedly loading models, which introduces negligible memory overhead.

Energy Saving. Figure 13 shows the end-to-end energy saving on real videos when targeting the same accuracy as the full model. Palleon achieves on average 4.9× energy saving (up to 6.7× on Good) compared to the full model. Palleon achieves a higher energy saving compared to "FAST (offline)" (*i.e.*, without counting the energy consumption in retraining) due to the early optimization strategy in ABLE. FAST (online) conducts model adaptation on the cloud and transfers the adapted model weights (in megabytes) to the edge through the network, leading to extra energy consumption from network communication. This network overhead becomes intensive when class skew switches frequently (Ocean and Friends), leading to more energy consumption in FAST (online) compared to the full model. This overhead reduces when class skews have longer lasting time (Departed and Good), leading to energy saving. By contrast, Palleon shows a consistent benefit on all four videos due to Palleon's low overhead.

Workload Distribution over Models. We observe that most frames are processed by the most compact model. For eaxmple, on "Friends" dataset, the most compact model processes 85% frames. While workload distribution varies for different CSPs, we observe similar trends across datasets. The reason is that CSPs usually only contain a few classes (Table 2) and the most compact model with Bayesian Filter can provide high accuracy.

6 Discussion

Comparison with alternative design that trains a model for each CSP. One alternative design to exploit class skews is to train many small models to recognize only a small set of

classes for individual class skews. This alternative design has two intrinsic drawbacks. First, we usually do not foreknow the class skew in an online video such that we can hardly train compact models for each class skew offline. Second, even if we assume that all class skews are foreknown (as the case in FAST), we may need to train a large number of models due to the large number of class skews. By contrast, Palleon does not assume that class skews are foreknown and trains the model on all classes offline. During online video analytics, we use ABLE to conduct online class skew detection, Bayesian Filter for efficient online model adaptation, and separability-aware model selection to automatically select CNNs for balancing the accuracy and resource efficiency.

Generality to other CNNs. Palleon can accelerate a large number of workloads on mobile devices with the temporal locality that a small number of classes keep appearing in a large number of consecutive frames. We have shown the performance benefits of Palleon on object classification and face recognition. Palleon can be generalized to 2D object detection [15] and 3D point cloud analytics [29] which share a similar pipeline as face recognition. We also note that Palleon can benefit from more compact models and pruning techniques designed for mobile systems. In particular, these compact models can be incorporated during model bank generation to provide Pareto-optimal boundary with reduced resource consumption and equivalent accuracy.

7 Related Work

Model Compression. Model compression has been widely explored for accelerating video processing. The popular compression techniques include resolution reduction [21, 37, 45, 58], matrix factorization [38, 61, 77], matrix pruning [14, 28], and distillation [8, 11, 13, 47]. Model compression is orthogonal to our work in exploiting class skews and usually leads to accuracy drop. By contrast, Palleon exploits class skews in video streams and maintains accuracy while reducing energy consumption and processing latency. Meanwhile, Palleon can integrate these compression techniques into our Separability-Aware Model Selection for generating compact models.

Video Processing with Low-Level Temporal Information. Using low-level temporal information can improve accuracy or reduce energy consumption. From the perspective of system design, existing work exploits low-level temporal information by caching processing results of the most recent frames for future computation reuse [26, 74] or adjusting sampling rate [41, 79]. From the perspective of algorithm design, existing work often augments the traditional 2D-CNN with optical flow [66, 71, 72] for explicitly capturing object motions across frames. A new CNN design, 3D-CNNs [12, 22, 69], has also been proposed to implicitly learn object motions by stacking several 2D-CNNs and processing adjacent video frames in a combined way. These works are orthogonal to our work because we focus on exploiting high-level temporal information across minutes, not on low-level temporal information in a few seconds. Palleon could be integrated with one of these approaches for further performance improvement.

Video Processing with High-Level Temporal Information. Several video processing systems [27, 32, 43, 65, 73, 75] have been proposed to exploit high-level temporal information across minutes, in terms of scenario information. Several early work [27, 32, 43, 75] simplifies processing tasks by targeting a specific scenario and only recognizing a specific object, *e.g.*, buses at a crosswalk. Recent work [65] conducts offline-profiling over a few scenarios and only reduces energy consumption when these offline-profiled scenarios appear, which would be in-effective for more realistic settings that class skews may switch during runtime. By contrast, Palleon abstracts these specific scenarios to a more general class skew of unbalanced distributions and enables online class skew detection and online model adaptation.

8 Conclusion

Efficient video processing on mobile platforms is an important workload. We present Palleon, a runtime system for efficient video processing, by detecting and exploiting class skews in video streams. We propose ABLE to detect class skews in video streams. Based on these detected class skews, Palleon uses Bayesian Filter for online model adaptation and Separability-Aware Model Selection to select the most energy-efficient model during runtime. Evaluations on both synthesized videos and real videos demonstrate that Palleon achieves up to 6.7× energy saving and up to 7.9× latency reduction. We conclude that Palleon is a highly practical and effective approach for efficiently processing video streams.

9 Acknowledgements

We would like to thank our shepherd, Swami Sundararaman, and the anonymous ATC reviewers. This work was supported in part by NSF 1925717 and 1725447.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, and Michael Isard. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[3] Awais Ahmad, Marco Anisetti, Ernesto Damiani, and Gwanggil Jeon. Special issue on real-time image and video processing in mobile embedded systems. *Journal of Real-Time Image Processing*, 16(1):1–4, Feb 2019.

[4] Advancing ai for video. https://phys.org/news/2019-06-advancing-ai-video-startup-powerful.html. Accessed: 2019-11-29.

[5] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 2017.

[6] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[7] Aria. The wearables giving computer vision to the blind. https://www.wired.com/story/wearables-for-the-blind/, 2018. Accessed: 2018-07-16.

[8] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NeurIPS*, 2014.

[9] BBC. Bbc series uses robot creatures to document secret lives of animals. https://www.theguardian.com/media/2016/dec/31/bbc-robot-creatures-spy-secret-lives-animals.-wildlife-series, 2018. Accessed: 2018-07-16.

[10] K. M. bin Saipullah, A. Anuar, N. A. binti Ismail, and Y. Soo. Real-time video processing using native programming on android platform. In *2012 IEEE 8th International Colloquium on Signal Processing and its Applications*, pages 276–281, March 2012.

[11] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, 2006.

[12] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017.

[13] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *NeurIPS*, 2017.

[14] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[15] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. In *NIPS*, pages 379–387, 2016.

[16] Dimitrios Dakopoulos and Nikolaos G Bourbakis. Wearable obstacle avoidance electronic travel aids for blind: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2009.

[17] Dell workstation t7910. https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell_Precision_Tower_7910_Spec_Sheet.pdf. Accessed: 2018-04-20.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[19] Paul Fearnhead and Zhen Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2007.

[20] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *NeurIPS*, 2016.

[21] Jianlong Fu, Heliang Zheng, and Tao Mei. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. In *CVPR*, 2017.

[22] L. Ge, H. Liang, J. Yuan, and D. Thalmann. 3d convolutional neural networks for efficient and robust hand pose estimation from single depth images. In *CVPR*, 2017.

[23] Ross Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1440–1448, USA, 2015. IEEE Computer Society.

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, page 580–587, USA, 2014. IEEE Computer Society.

[25] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[26] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *ASPLOS*, pages 271–284, 2018.

[27] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *MobiSys*, 2016.

[28] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[29] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-Sheng Hua, and Lei Zhang. Structure aware single-stage 3d object detection from point cloud. In *CVPR*, pages 11870–11879. IEEE, 2020.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[31] Marion Hersh and Michael A Johnson. *Assistive technology for visually impaired and blind people*. Springer Science Business Media, 2010.

[32] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, 2018.

[33] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[34] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size, 2016.

[35] Spectrum IEEE. Robot takes on landmine detection while humans stay very very far away. https://spectrum.ieee.org/automaton/robotics/military-robots/husky-robot-takes-on-landmine-detection-while.-humans-stay-very-very-far-away, 2018. Accessed: 2018-07-16.

[36] iphone x specification. https://www.gsmarena.com/apple_iphone_x-8858.php. Accessed: 2019-12-3.

[37] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018.

[38] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[39] Theodoros Damoulas Jeremias Knoblauch. Spatio-temporal Bayesian on-line changepoint detection with model selection. In *ICML*, 2018.

[40] Jetson nano specification. https://developer.nvidia.com/embedded/buy /jetson-nano-devkit. Accessed: 2018-04-20.

[41] Angela H. Jiang, Daniel L.-K. Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A. Kozuch, Padmanabhan Pillai, David G. Andersen, and Gregory R. Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *ATC*, 2018.

[42] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 253–266, New York, NY, USA, 2018. ACM.

[43] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *VLDB*, 2017.

[44] David G Kleinbaum and Mitchel Klein. *Survival analysis*, volume 3. Springer, 2010.

[45] A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 1, 01 2009.

[46] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191, 2017.

[47] D. Lopez-Paz, B. Schölkopf, L. Bottou, and V. Vapnik. Unifying distillation and privileged information. In *ICLR*, 2016.

[48] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[49] Othmane Mazhar, Cristian Rojas, Carlo Fischione, and edit Mohammad Reza Hesamzadeh. Bayesian model selection for change point detection and clustering. In *ICML*, 2018.

[50] Extech multimeter model ex330 manual. http://www.extech.com/resources/EX330_UM.pdf. Accessed: 2018-04-20.

[51] Build a hardware-based face recognition system for $150 with the nvidia jetson nano and python. https://medium.com/@ageitgey/build-a-hardware-based-face-recognition-system-for-150-with-the-nvidia-jetson-nano-and-python-a25cb8c891fd. Accessed: 2019-06-12.

[52] Home automation at a glance using ai glasses. https://hackaday.com/2019/08/15/home-automation-at-a-glance-using-ai-glasses/. Accessed: 2019-06-12.

[53] Jetbot, a $250 diy autonomous robot based on jetson nano impresses at gtc. https://blogs.nvidia.com/blog/2019/03/26/jetbot-diy-autonomous-robot/. Accessed: 2019-06-12.

[54] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *ICML*, 2005.

[55] Wikipedia: Pareto efficiency. https://en.wikipedia.org/wiki/Pareto _efficiency. Accessed: 2019-06-12.

[56] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[57] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *BMVC*, 2015.

[58] Matthai Philipose. Efficient object detection via adaptive online selection of sensor-array elements. In *AAAI*, 2014.

[59] Raspberry pi 3b+ specification. https://www.raspberrypi.org/products /raspberry-pi-3-model-b-plus/. Accessed: 2018-04-20.

[60] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[61] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[62] W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976.

[63] Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *ICML*, 2010.

[64] J. Shao. *Mathematical Statistics*. Springer Texts in Statistics. Springer, 2003.

[65] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *CVPR*, 2017.

[66] Zheng Shou, Xudong Lin, Yannis Kalantidis, Laura Sevilla-Lara, Marcus Rohrbach, Shih-Fu Chang, and Zhicheng Yan. Dmc-net: Generating discriminative motion cues for fast compressed video action recognition. In *CVPR*, 2019.

[67] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[68] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018.

[69] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[70] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[71] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Val Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.

[72] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *CVPR*, 2018.

[73] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *WWW*, 2019.

[74] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *MobiCom*, 2018.

[75] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In *EuroSys*, 2019.

[76] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[77] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. In *ICASSP*, 2014.

[78] Aonan Zhang and John Paisley. Deep Bayesian nonparametric tracking. In *ICML*, 2018.

[79] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, 2017.

[80] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*, pages 7543–7552, 2019.