Close the Gap between Deep Learning and Mobile Intelligence by Incorporating Training in the Loop

Cong Wang
Old Dominion University
Norfolk, VA
c1wang@odu.edu

Yanru Xiao
Old Dominion University
Norfolk, VA
yxiao002@odu.edu

Xing Gao
University of Memphis
Memphis, TN
xgao1@memphis.edu

Li Li
Chinese Academy of Science
Shenzhen, China
li.li@siat.ac.cn

Jun Wang
Futurewei Technologies
Santa Clara, CA
jun.wang2@huawei.com

ABSTRACT

Pre-trained deep learning models can be deployed on mobile devices to conduct inference. However, they are usually not updated thereafter. In this paper, we take a step further to incorporate training deep neural networks on battery-powered mobile devices and overcome the difficulties from the lack of labeled data. We design and implement a new framework to enlarge sample space via data paring and learn a deep metric under the privacy, memory and computational constraints. A case study of deep behavioral authentication is conducted. Our experiments demonstrate accuracy over 95% on three public datasets, a sheer 15% gain from traditional multi-class classification with less data and robustness against brute-force attacks with 99% success. We demonstrate the training performance on various smartphone models, where training 100 epochs takes less than 10 mins and can be boosted 3-5 times with feature transfer. We also profile memory, energy and computational overhead. Our results indicate that training consumes lower energy than watching videos so can be scheduled intermittently on mobile devices.

CCS CONCEPTS

• **Human-centered computing** → *Mobile computing*; • **Security and privacy** → **Biometrics**; • **Computing methodologies** → *Neural networks*.

KEYWORDS

On-device machine learning; privacy preservation; deep metric learning; behavioral authentication

MM '19, October 21–25, 2019, Nice, France

ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3350904

ACM Reference Format:
Cong Wang, Yanru Xiao, Xing Gao, Li Li, and Jun Wang. 2019. Close the Gap between Deep Learning and Mobile Intelligence by Incorporating Training in the Loop. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19), Oct. 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3350904

1 INTRODUCTION

Being close to the data source, smartphone is one of the ideal platforms for deep learning applications. Rather than querying cloud servers with private data, users can compute from a local model, hence sensitive information never leaves their devices. By relaxing the communication requirements to the cloud, network bandwidth and service charges can be saved. Users thus enjoy better experience, especially in many real-time applications such as object detection from video streams. Early endeavors either explore new variants of applications such as activity recognition [1, 17] and mobile vision [21, 37], or optimize the performance with model compression [6] and distillation [2, 9]. Their goal is to improve computational efficiency on resource-constrained mobile devices.

Most of the existing schemes only consider *inference* on mobile from a pre-trained model. Unfortunately, being able to infer from a static model still leaves a significant gap from being cognizant. Machine learning relies on the assumptions that the test samples are independently and identically drawn from the same distribution at the training time. Deep classifiers are not good at extrapolation when the data comes from a different distribution. However, it is quite common in mobile applications. Take behaviorial authentication [10, 18, 23, 27, 28, 36, 40] and activity recognition [1, 17] for example, behaviorial patterns may evolve due to sickness, injury and emotion, thus intensify the intra-class variations and hamper classification. As a result, the model should be re-trained or finetuned constantly to adapt to the new data distribution. Therefore, closing the gap between deep learning and mobile intelligence requires to bring training back into the loop.

The existing solutions include: 1) hosting an enclave for each client in the cloud [31], securely aggregating data from

users, keeping all the models updated and providing downlink accessibility whenever requested; 2) using homomorphic encryptions to obfuscate both private data and model, while the cloud can still conduct meaningful computations on encrypted data or model weights [12, 38]. These approaches are obviously not scalable for large-scale systems. Other solutions project the data into non-sensitive representations [11, 20], but do not offer rigorous security guarantees and often come with a risk of accuracy loss.

Motivated by the latest advances in mobile processors, in this paper, we take a step further to incorporate both *training* and *inference* (i.e., the entire loop of learning) on mobile devices. Specifically, we focus on the challenge to perform learning effectively on resource-constrained mobile devices, and the associated cost. In contrast to the cloud environments, where large amount of labeled data has been collected offline, a fundamental challenge on user's mobile is the lack of *labeled data*, since interactive labeling impacts user experience. Directly learning from small data sets leads to overfitting and degrades performance. Thus, the challenge comes from a multi-dimensional design space considering *accuracy*, *privacy* and *performance*.

This paper presents a first exploration to tackle this multi-faceted challenge. For *accuracy*, we make samples into pairs and learn a deep metric to mitigate overfitting [3, 15]. For *performance*, we implement feature transfer to speed up training convergence on mobile, while securing all the intermediate activations/model parameters. We further develop a space-time decision fusion algorithm to enhance the reliability of decisions in dynamic mobile environments. The inference results are fed back to schedule model training hence close the loop and preserve the *privacy* of user data. Finally, we conduct a comprehensive use case study of deep behavioral authentication, that takes gait modality for implicit authentication with minimum impact on user experience.

The main contributions are: 1) we conduct both training and inference on mobile devices to preserve privacy and learn effectively in a dynamic mobile environment. Our implementation demonstrates that training is not only feasible on mobile but also quite fast with feature transfer (within 5s/epoch on Huawei Mate10 for 400 samples); 2) we tackle potential overfitting by paring data samples under the memory constraint and learn a deep metric to enhance the discriminative power of the model. In our case study, the experiments demonstrate 10-15% increase of authentication accuracy on different datasets and achieve an accuracy of 0.94 on a large dataset with 153 participants; 3) we profile performance of different model architectures and memory/energy cost on various smartphone models. To the best of our knowledge, this is the first work that implements both training and inference, and addresses the associated challenges on battery-powered mobile devices.

The rest of the paper is organized as follows. Section 2 presents the background knowledge and motivation. Section 3 describes the system design. Section 4 conducts a case study to evaluate the framework and Section 5 concludes this work.

2 RELATED WORKS

2.1 On-Device Deep Learning

Deep learning frameworks can be exported to mobile for inference. Yet, training consists of forward and backpropagations. The forward propagation resembles the inference except that the intermediate gradient values are stored in memory. In backpropagation, the prediction error is backpropagated through the network and the model parameters are adjusted towards minimizing a loss objective (using the gradients). Although some works have evaluated embedded platforms with proprietary GPU (e.g. Nvidia Tegra series) [7, 19], these platforms have no essential difference from the cloud setting (Ubuntu, CUDA and external power). In contrast, we consider *battery-powered* smartphones running Android with multi-core CPUs that are thermally and electrically limited. Since training consumes additional memory, for mobile applications, most of the existing frameworks (e.g. *Tensorflow Lite, Caffe2, MXNet*[1]) have tailored backpropagation and left only with the inference part to compute from pre-trained models [35].

The previous works mainly focus on model compression for inference [2, 5, 6, 9]. Quantization is a typical method to compress the model, that rounds the original 32-bit floating point parameters into the 8-bit integer with 75% model size reduction [5]. It is desirable for inference in one-shot computation, whereas training still requires high accuracy especially for many security-critical applications. In [6], a pre-trained model is pruned to have sparse connections. In [2, 9], a shallow model is trained to learn complex functions from a deep model. These works are useful at the deployment stage once the model has been trained, but where and how to train the deep model are not considered.

2.2 Behavioral Authentication

Smartphone features a variety of sensors to capture behavioral information using acceleration, gyroscope, etc. Behavioral biometrics such as gait [10, 27, 36], screen touch [40], keystroke dynamics [23, 28] and eye movement [18] are proven to be successful in differentiating human subjects. They reflect the internal characteristics of a user, and are difficult to replicate. A system process can run continuously in the background for implicit authentication with no deliberate attention from the user [14], which makes behavioral biometrics an ideal *second factor* for authentication. Based on statistical features, the previous works focus on using deterministic algorithms or classifiers with less discriminative power [10, 27, 36], whereas data outliers, abrupt changes could easily mislead these techniques.

Convolutional Neural Networks (ConvNets) are used in [4] to extract features from the pre-processed sensing signals to recognize users. Similarly, a recurrent neural network is constructed to learn continuous motion patterns [24]. They train a homogeneous model for all users on cloud servers. However,

[1] Tensorflow Lite, https://bit.ly/32flUV8
Caffe2 for iOS/Android, https://bit.ly/2SGx6pP
MXNet for smart devices, https://bit.ly/2Su7axk

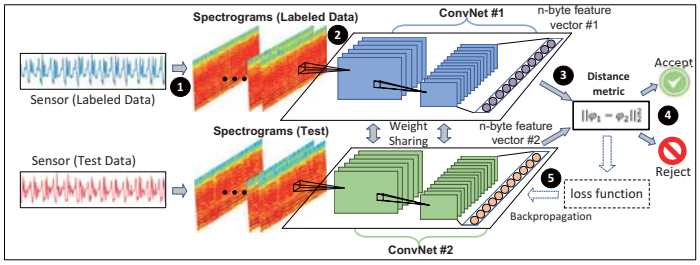


Figure 1: System architecture on mobile devices.

Algorithm 1: Memory-efficient Sampling

Input: r^2 positive and $n_s rs$ negative pairs, memory bound $2R$.
Output: a balanced set of samples of size $2R$.
Set of all negative pairs $\mathcal{N}$, $R = r^2$, $|\mathcal{N}| = n_s rs$.
for $T \leftarrow 1, \cdots, R$ **do**
  $\mathcal{R} \leftarrow \mathcal{R} + (i \in \mathcal{N})$.
for $T \leftarrow R+1, \cdots, n_s rs$ **do**
  if *probability* $p > \frac{R}{T}$ **then**
    $\mathcal{R} \leftarrow \mathcal{R} - (i \in \mathcal{R}) + (i \in \mathcal{N})$.

user's biometric data is private and may be improperly used by curious cloud vendors to infer health or mental condition, e.g. predicting Parkinson's disease, depression from gait patterns [22, 29]. Such privacy issue is tackled by training on the mobile devices discussed next.

3 FRAMEWORK DESIGN

The framework is depicted in Fig. 1: ❶ It takes raw sensor inputs, transforms them into mid-level representations (spectrograms [8]). The benefit of this transformation is evaluated in Section 4.2.1; ❷ processes the representations with the neural network; ❸ computes a distance metric from the feature vectors; ❹ generates a decision; ❺ backpropagates the error if training is scheduled.

3.1 Deep Metric Learning

Although sensing data is abundant on smartphone, labeling requires external efforts and bootstrapping time from the user. Existing approaches of multi-class classification typically use the softmax function to output a probability for each predicted class. For a total of k classes and n samples in each class, it learns from the $\mathcal{O}(kn)$ samples. When n is small, the model is subject to overfitting with softmax. A solution is to pair the samples and learn a similarity distance metric using the *siamese network* [3, 15]. This way, the input is expanded by a factor of kn. As shown in Fig. 1, it incorporates two branches of identical ConvNets that share model weights. They take a series of convolution, nonlinear activation and downsampling to yield feature vectors φ_1, φ_2, and merge into a top network to learn a distance metric function $f(\varphi_1, \varphi_2)$. It is constructed with the *contrastive loss* function to map feature vectors to a space in which similar samples have closer distance whereas dissimilar samples are far apart (separated by a margin). For a pair i, j of dataset $\mathcal{D}$, the contrastive loss function is defined as,

$$\mathcal{L}_c = \sum\nolimits_{i,j \in \mathcal{D}} y(\varphi_1^{(i)}, \varphi_2^{(j)}) f(\varphi_1^{(i)}, \varphi_2^{(j)})^2 + (1 - y(\varphi_1^{(i)}, \varphi_2^{(j)})) \max(m - f(\varphi_1^{(i)}, \varphi_2^{(j)}), 0)^2, \quad (1)$$

in which label $y(\varphi_1^{(i)}, \varphi_2^{(j)}) = 0$ for dissimilar pairs and $y(\varphi_1^{(i)}, \varphi_2^{(j)}) = 1$ for similar pairs. m is the margin. If the pair is similar (positive), the loss is $f(\varphi_1^{(i)}, \varphi_2^{(j)})^2$; if the pair is dissimilar (negative), the loss is $\max(m - f(\varphi_1, \varphi_2))^2$. When $f(\varphi_1, \varphi_2) > m$, the loss is zero, i.e., dissimilar pair with distance larger than the margin has zero loss. A slightly different loss function is proposed in [15], that does not require the margin parameter. We evaluate both functions in Section 4.

Memory-efficient Sampling. Training takes batched input in memory sampled from flash storage. To avoid the latency accessing the storage, the system maintains a pool of sampled pairs in memory. This makes sampling crucial because of data balance and increased memory footprint. Consider authentication as an extreme case, where the number of negative samples is much larger than the positive ones (from the device owner). Denote variables of n_s negative classes of s samples (supplied by the cloud as discussed next). For the mobile user with r samples, there are r^2 positive pairs and the $n_s rs$ negative pairs ($n_s rs \gg r^2$). Since loading all the negative pairs into memory may lead to memory leaks, the goal is to keep a random subset of negative samples within memory limits.

We develop a balanced reservoir sampling algorithm. A buffer size of $2R$ is found from hardware configuration or test (half for positive and half for negative pairs). The size determines a trade-off between memory usage and variety of negative records. Small R could lead to severe overfitting and large R risks of having memory error. To maximize coverage, we set $R = r^2$ so all positive samples are utilized for training and make sure that the total size of $2R$ is within the memory capacity. The algorithm continuously adds record into the reservoir till the $(T+1)$-th record, $T = R$. If $T > R$, a random pair in the reservoir is replaced with probability $\frac{R}{T}$ or rejected with probability $1 - \frac{R}{T}$. After the sequential pass through all the records, the buffer forms a random set from the pool of negative samples.

3.2 Decision Fusion and Feedback

After the model is trained, the inference module takes input from sensors and outputs a classification decision. The decision based on a single shot of inference is not reliable because interference, outliers, and behavioral instability persist at run-time. The goal is to reach a high confidence within minimum observation time. We build an algorithm on top of the inference module to fuse multiple inferences across spatial and temporal axes. For data x_i at time i, we first perform spatial selections from the training samples. x_i is paired with k samples randomly selected from the training set on mobile, since one training sample is not sufficiently representative. The mean distance d_i from k random samples $d_i = \sum_{j=1}^{k} d(x_j, x_i)/k$ is computed.

Not only could the selection of training samples have imperfections, the incoming data may also have disturbances. After the spatial evaluation, we progress along the time dimension to fuse multiple decisions $\{y_1, y_2, \cdots, y_n\}$. After the i-th evaluation, it either decides to *accept* (H_0), *reject* (H_1) or *continue* to observe y_{n+1}. The module defines two kinds of errors: false negative α and false positive β. The objective is to minimize the expected time of evaluation and satisfy the error constraints, which is formulated as Sequential Probability Ratio Test (SPRT) [33]. SPRT progresses by assessing a likelihood ratio λ_n for the n-th observation,

$$\lambda_n = \frac{p(y_1, \cdots, y_n|H_1)}{p(y_1, \cdots, y_n|H_0)} = \prod_{i=1}^{n} \frac{p(y_i|H_1)}{p(y_i|H_0)}. \tag{2}$$

The second equality holds because samples are independently randomly drawn. We extend SPRT for the distance metric (contrastive loss). Pairs with distance less than the margin threshold (typically set to $m/2$) are considered as similar; otherwise, they are dissimilar. We use a normal distribution to model the distance into probability distribution,

$$p(d_i|\mu, \sigma^2) = 1 - \phi(\frac{d_i - \mu}{\sigma^2}), \tag{3}$$

in which (μ, σ^2) is set to $(\frac{m}{2}, 0.25)$ in the experiment. Distance around 0 or margin m has high probability being similar or dissimilar, and lower probability around $\frac{m}{2}$ when the classifier is unsure. Combining (2) and (3), the ratio is,

$$\frac{p(y_i = 0|H_1)}{p(y_i = 0|H_0)} = \phi(\frac{d_i - \mu}{\sigma^2})/\big(1 - \phi(\frac{d_i - \mu}{\sigma^2})\big) \tag{4}$$

$$\frac{p(y_i = 1|H_1)}{p(y_i = 1|H_0)} = \big(1 - \phi(\frac{d_i - \mu}{\sigma^2})\big)/\phi(\frac{d_i - \mu}{\sigma^2}) \tag{5}$$

The strategy is proven to be optimal if the following decision is made,

$$S_n^* = \begin{cases} H_0, \lambda_n \leq B \\ H_1, \lambda_n \geq A \\ \textit{continue}, B < \lambda_n < A \end{cases} \tag{6}$$

We set the two thresholds A and B suggested by [33], $A = (1 - \beta)/\alpha$, $B = \beta/(1 - \alpha)$. The sequence moves within the open interval (B, A) till a decision is made. Intuitively, if consecutive decisions of acceptance are made, the likelihood ratio shrinks multiplicatively. Any rejection along the way would drive the ratio to an opposite direction towards the upper threshold until a threshold is met. The decision of S_n^* is examined closely to schedule training.

Feedback. We examine the testing accuracy as a feedback to schedule model re-training to adapt variations. Again, take authentication as an example, if the decision outputs a false negative, the screen is mistakenly locked by the (second-factor) behavioral authentication, but the user later logins with her face or fingerprint (that verifies the decision is indeed a false negative). If such situations exceed a certain number, it indicates that the user's behavior may have undergone a substantial change and training is scheduled with a mix from the new data. Incorporating training on mobile could immediately respond to these shifts thereby closing the loop of learning on mobile devices. The scheme is summarized in Algorithm 2 and evaluated in Section 4.2.6.

Algorithm 2: Decision Fusion and Feedback

Input: Testing pairs (x_j, x_i), $1 \leq j \leq k$. k pairs randomly drawn from training set. False negative α and false positive β, threshold $A = (1 - \beta)/\alpha$, $B = \beta/(1 - \alpha)$.
Output: Decision S_n^* and training schedules.
Initialize false negative counter $c \leftarrow 0$, and threshold T.
while $c < T$ **do**
  $n \leftarrow 0$
  while $B < \lambda_n < A$ **do**
    $\overline{d_i} \leftarrow \sum_{j=1}^{k} d(x_j, x_i)/k, p(d_i|\mu, \sigma^2) \leftarrow 1 - \phi(\frac{d_i - \mu}{\sigma^2})$.
    $\lambda_n \leftarrow \prod_{i=1}^{n} \frac{p(d_i|H_1)}{p(d_i|H_0)}$.
    if $\lambda_n \geq B$ **then**
      $S_n^* \leftarrow 1$ and **Break**.
    if $\lambda_n \leq A$ **then**
      $S_n^* \leftarrow 0$ and **Break**.
    $n \leftarrow n + 1$
  Output optimal decision S_n^*.
  if *Given true label* H_0, $S_n^* = H_1$. **then**
    $c \leftarrow c + 1$
Schedule training of $\mathcal{M}_t$ with new data $\mathcal{D}_t$.

4 USE CASE STUDY OF BEHAVIORAL AUTHENTICATION

To evaluate the framework, we conduct a case study of behavioral authentication using *gait* (walking) data collected by the accelerometer sensors [10, 27, 36]. The primary goal of the evaluation is to defend random attackers who may use their own data or samples from a large database to spoof the authentication system. This case is common since a random attacker may obtain a device lost by the victim. Without any prior knowledge on the behavioral pattern, he can retrieve data from a large public database and launch brute-force attacks to unlock the device. Meanwhile, the classifier should be able to recognize the device owner across various sessions though behavioral patterns may change significantly.

4.1 Model and Mobile Development

We convert the tri-axial accelerometer signal into spectrograms [8] and stack them vertically. Spectrogram encodes the time serial data into 2D images of time-frequency representation. This way, learning can be performed effectively using convolutional neural networks[2].We evaluate three model architectures extended from *LeNet* [16], *VGG* [32] and *MobileNetv2* [30] by adding or pruning layers to yield similar input dimension at the dense layer as their original implementations. We develop the system on a Java-based framework called *DL4J*[3] and enable SIMD to use multi-core CPUs. During testing, we notice that deeper structures could cause `OutOfMemoryError` due to a large number of parameters and batched data processing. To mitigate, we set `largeHeap` to give the application a 512 MB heap capacity and select a sampling buffer size less than this value.

[2]Though an option is to use the recurrent neural networks, their computation intensity are much higher on the mobile devices.
[3]Deep Learning for Java, https://deeplearning4j.org

		baseline			siamese multi-class		siamese binary-class (20% data)	
		softmax(sw)	softmax(spgm)	osvm	contrastive	cross-entropy	contrastive	cross-entropy
Mcgill	LeNet4	0.774	0.881	0.542	0.918	0.940	**0.966**	0.934
	VGG8	0.752	0.902	0.672	0.925	0.952	**0.962**	0.906
	MobileNetv2	0.682	0.811	0.581	0.865	**0.926**	0.847	0.901
IDNet	LeNet4	0.726	0.842	0.552	0.884	0.903	**0.937**	0.899
	VGG8	0.764	0.875	0.561	0.916	0.908	**0.934**	0.901
	MobileNetv2	0.770	0.776	0.591	0.876	0.912	0.910	**0.921**
ZJU	LeNet4	0.442	0.646	0.511	0.681	0.804	**0.941**	0.926
	VGG8	0.463	0.743	0.523	0.769	0.841	**0.936**	0.851
	MobileNetv2	0.591	0.471	0.510	0.706	0.778	**0.895**	0.835

Table 1: Model accuracy of different loss functions and network architectures

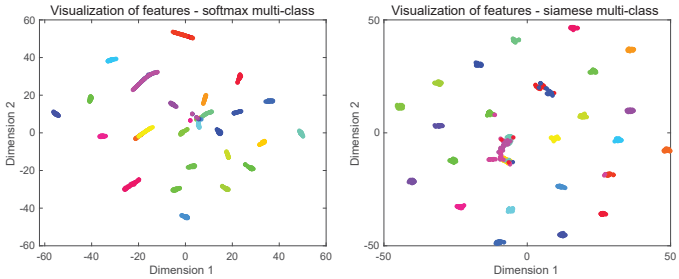
4.2 Experiments

The main goals of the experiments are: 1) investigate the accuracy and computational cost of different models and approaches; 2) examine cost savings and performance impact from feature transfer; 3) validate system robustness against intra-class variations and random attacks; 4) profile performance and overhead on various smartphone models.

Datasets. To make the benchmarks comparable, the experiments are based on public datasets: Mcgill [13], IDNet [4], ZJU [39] and Osaka [25] gait datasets. With a total coverage of around 1,000 individuals, we believe the four datasets are sufficient to validate the system in various scenarios. In particular, Mcgill includes 15-min walk of 20 people. IDNet is collected with different types of phones and dresses from 50 people. ZJU contains 153 individuals in 3 sessions with 5 body sensors. Osaka records 1-minute walk of 744 subjects. Due to short recordings (only 1-2 spectrograms), we cannot perform meaningful training even with sample paring so it is utilized as a large database from which attackers may launch random attacks.

The datasets are split into 80% for training and 20% for testing. The test set is generated by randomly pairing training samples with testing samples. This simulates the run-time when new motion data is evaluated against training samples as the ground truths captured during the bootstrap phase. To assess the performance of authentication, we mainly focus on the mean Average Precision (mAP), which is the average percentage of true authentication over the total number of testing. We set the margin $m = 1.5$ in the contrastive loss (Eq. (1)). For fast prototyping, we first develop the model and evaluate authentication accuracy, security and performance in Tensorflow with Nvidia Tesla P100 GPU, and then develop the learning module on Nexus 6/6P, Huawei Mate 10 and Google Pixel2 using DL4J. During our testing, we find that the maximum batch size for Nexus 6 (oldest phone in our test) is 56 (pairs). To test various models and avoid memory errors, we set the batch size to 20 on mobile.

4.2.1 Accuracy. We first evaluate authentication accuracy, compare models, data representation and learning mechanisms on different datasets and examine the gap between multi-class and binary classifications (Table 1). First, we show that data representation has a significant impact on accuracy. Existing research mainly works in the time-domain



(a) (b)

Figure 2: Softmax vs. siamese classification via t-SNE visualizations (a) softmax; (b) siamese.

to extract cycle [4] or segment temporal signals [26]. The first two columns of the table compares spectrogram representation with sliding window (SW) on the temporal signal [26]. Spectrogram achieves a significant accuracy gain of over 10%. A one-class SVM (osvm) is used in [4] to detect outliers from imposters. Our experiments show that, although osvm can handle 80-90% outliers, it fails to generalize to the positive samples, which results in high rate of false rejections.

The siamese network can be used for both multi-class and binary classifications. Multi-class classification requires all the pairs between different classes to be labeled [4, 24] whereas binary only labels one vs. the rest. To simulate limited mobile storage, only 20% data from the training set is used for binary classification but tested on the entire test set. This is challenging for recognition since the ConvNet can only "see" from a small subset of training data. A model is trained for each individual in the dataset and the results are averaged. First, it is observed that the siamese network improves accuracy significantly. Compared (column 4/5) to softmax (column 2), multi-class siamese offers 8-15% accuracy gain. We visualize the features learned by softmax and siamese (contrastive loss) in Fig. 2(a) and (b), where the colors represent the feature vectors of different subjects in 2D. Features learned by softmax are not sufficiently discriminative where the distance along the feature vectors from the same individual could be similar to a different individual. We further notice that some features belong to different individuals are mapped to the same vector space in 2D. These findings are in line with [34] (softmax tends to underperform). Contrastive loss from the siamese network offers improvements by mapping

feature activations into a condensed, compact set of spaces. This shows the higher discriminative power of deep metric learning than softmax especially with less training data.

However, accuracy still declines a little with an increasing number of classes (e.g. from 0.952 of Mcgill with 20 people down to 0.841 of ZJU with 136 people). This problem is tackled by only doing binary classification, which is quite reasonable on mobile. As observed, the accuracy stays above 90%. We also notice some interesting phenomenon that the cross-entropy loss is better than the contrastive loss for multi-class classification, but the opposite for binary classification. The difference between them is that the cross-entropy generates a probabilistic decision, rather than a deterministic distance metric from the contrastive loss. In our experiment, we discover that contrastive loss is more prone to error during multi-class classification in the presence of hard samples. Due to space limit, we plan to conduct more experiments in our future work. Finally, we further alter the model into VGG8 and MobileNetv2. VGG8 achieves the best accuracy in most cases. With 40% less parameters, MobileNetv2 suffers 8-26% accuracy loss compared to LeNet4.

4.2.2 Resource Requirement. Fig. 3 illustrates the relations between model parameters, floating point operations (FLOPS), and accuracy. We alter the structures by shrinking/expanding filter size, numbers, and adding/removing convolutional or pooling layers. For the same model, in general, more parameters bring higher representational power at the risk of overfitting and cost of computation. From Fig. 3(a), VGG8 is more stable than others in terms of accuracy. Once the number of parameters exceeds a million, the models tend to overfit. Mobilenetv2 can be tailored to only weigh half of LeNet4, but the performance is not stable. Fig. 3(b) also indicates that it incurs nontrivial GPU time if the FLOPS increase. Fig. 3(c) shows that LeNet4/VGG8 are more competitive than Mobilenetv2 for the datasets in terms of computation time and accuracy.

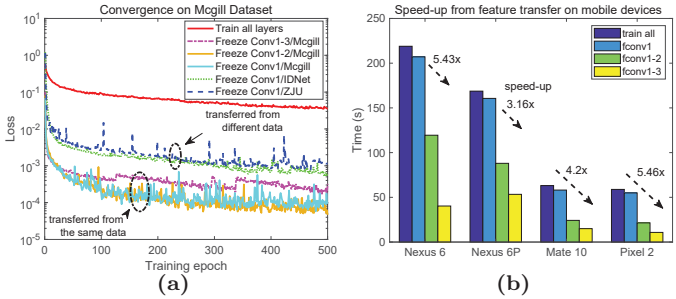
To facilitate mobile development, we conduct the following experiments using LeNet4 and keep the consistency through the rest of the experiments. Fig. 3(d) shows the training time per epoch on mobile devices. We plot in 3D for better visualization of the impact from the convolutional and dense layer. Training on mobile devices is not only feasible, but actually much faster than expected. For a deep model with 650K parameters and 400 samples, it only takes the latest Pixel2 or Mate10 less than 5 seconds to complete one training epoch. Thus, training 100 epochs takes less than 10 mins. Even the old Nexus 6 finishes around 10 seconds per epoch. During the experiment, we notice that the speed bottleneck of convolutional layers is magnified on mobile devices due to less processing power from the mobile CPUs and memory. As observed in Fig. 3(d), with more convolutional layers, training time surges sharply. However, increasing computations of the dense layer has less impact on performance. Interestingly, we are even able to train some networks with over a million parameters, as long as most of the parameters reside in the dense layer. Equipped with the capability to learn, model updates can be scheduled efficiently without external efforts from service providers.

4.2.3 Speed up on Mobile by Feature Transfer. Since convolutional layers learn common features, these features can be efficiently transferred from the cloud for computation efficiency. To see such potential, the following cases are evaluated: 1) freeze all convolutional layer weights (*fconv1-3*); 2) freeze first two convolutional layer weights (*fconv1-2*); 3) freeze the first convolutional layer weights (*fconv1*). We train the rest of the layers. The source model conducts multi-class classification on the dataset (public) without the presence of the target user (private). At the target user, it performs the binary classification based on the weights transferred from the source model. Note that this implementation is robust against privacy exploits since the private activations are kept on mobile and the transferred features are public. We also evaluate scenarios when different public data are available, by alternating the source data between the other two datasets. This allows us to examine the generality of features and their impact on accuracy and convergence.

Fig. 4(a) shows the convergence of a random individual from the Mcgill dataset. We can see that feature transfer offers at least two orders of magnitude speed-up in terms of convergence. Features learned from data gathered with different settings offer significant boost as well. For instance, for the loss value to converge to 0.05, the original training

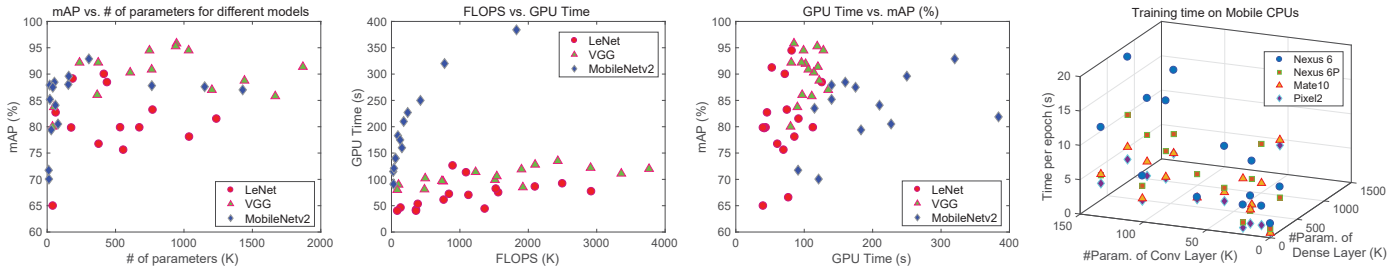
feature transfer		Mcgill	IDNet	ZJU	gain/loss
Mcgill	*fconv1-3*	0.933	0.903	0.907	-5.2%
	fconv1-2	0.948	0.927	0.918	-3.5%
	fconv1	0.953	0.941	0.948	-1.9%
	gain/loss	-2.1%	-4.2%	-4.2%	–
IDNet	*fconv1-3*	0.876	0.941	0.896	-3.3%
	fconv1-2	0.922	0.951	0.911	-0.9%
	fconv1	0.933	0.957	0.936	+0.5%
	gain/loss	-2.7%	+1.3%	-2.3%	–
ZJU	*fconv1-3*	0.808	0.810	0.829	-12.5%
	fconv1-2	0.836	0.818	0.833	-11.3%
	fconv1	0.832	0.804	0.847	-11.3%
	gain/loss	-11.6%	-13.0%	-10.5%	–

Table 2: Accuracy with feature transfer



(a) (b)

Figure 4: Boost from feature transfer (a) speed of convergence; (b) speed-up on mobile devices.



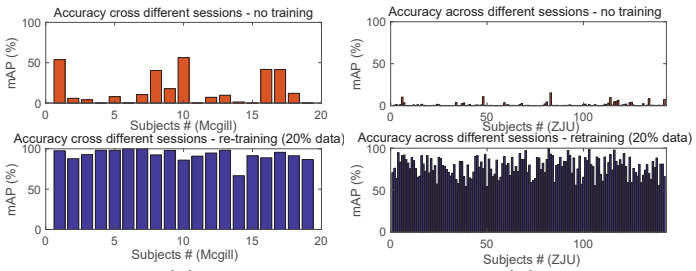
(a) (b) (c) (d)

Figure 3: Evaluation of resource requirement vs. accuracy on GPU and mobile platforms using IDNet (a) mAP vs. parameters; (b) FLOPS vs. GPU time; (c) GPU time vs. mAP; (d) Parameters (Conv and Dense Layers) vs. Mobile CPU Time.

takes 325 epochs. With feature transfer, it only takes 2 epochs from the same dataset, 5 and 4 epochs for different IDNet and ZJU datasets, respectively. We then evaluate the speed-up on mobile devices and measure the total computation time to finish 50 epochs of training, as shown in Fig. 4(b). Freezing all the convolutional layers offers 3-5 times of speed-up. If one additional convolutional layer is released, the gain is still over 2 times. The speed-up comes with a little accuracy loss due to the discrepancy among domain features (illustrated in Table 2). Training the dense layers only has 3-5% accuracy loss on Mcgill, IDNet, and 12% on ZJU dataset. The accuracy can be improved by fine-tuning more layers (e.g. to 0.9% and 3.5% for Mcgill and IDNet). Transferring from a different dataset only incurs minor accuracy loss (1-3% on average). This indicates that the proposed architecture is robust to re-use features for the new target domain, though device settings such as sampling frequency (sensors) can be different.

4.2.4 Robustness against Intra-class Variations. We show that scheduled training can adapt to intra-class variation when behavioral biometrics evolve. We utilize Mcgill and ZJU datasets since they record more than two sessions of a subject on different days (Mcgill) and months (ZJU). To see whether the system can still recognize its owner, we examine the acceptance rate. If the acceptance rate is low, the model is likely to reject the genuine user and degrade usability significantly. In the upper figures (*no training*) of Fig. 5, each user trains a model in session 1 and directly tests on the data from session 2. As we observe, the acceptance rate is quite low if the model is not updated. Mcgill dataset across several days only yields 16.3% average acceptance, and the rate drops to 1.1% for ZJU over a longer period. It certainly indicates that pre-trained models cannot adapt to new data distributions.

With continuous model updates, we fine-tune the model from the previous weights with a lower learning rate, and only use 20% of the new data. The bottom figures in Fig. 5 shows the mean acceptance percentage over all fine-tuning epochs, which quickly brings it back to 92.4% and 77.6% for Mcgill and ZJU, respectively. The best acceptance percentage of some users can hit 100% indicating that the fine-tuned model can almost perfectly adapt to the new data.



(a) (b)

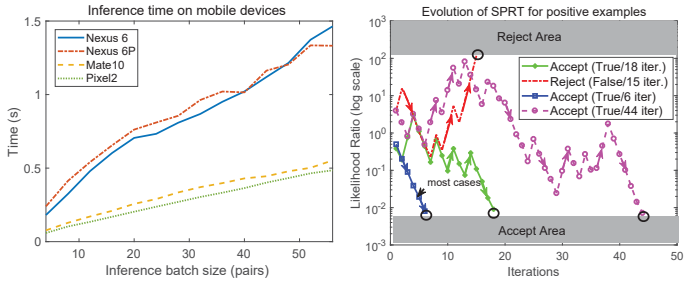
Figure 5: Acceptance rate across different sessions (a) Mcgill; (b) ZJU.

Dataset	All	Batch 4	Batch 8	Batch 16	Batch 32
Mcgill	0.05%	0.003%	0.003%	0.000%	0.000%
IDNet	2.36%	2.18%	2.014%	1.682%	1.024%
ZJU	0.346%	0.028%	0.010%	0.004%	0.001%

Table 3: Success ratio of random attacks using Osaka dataset

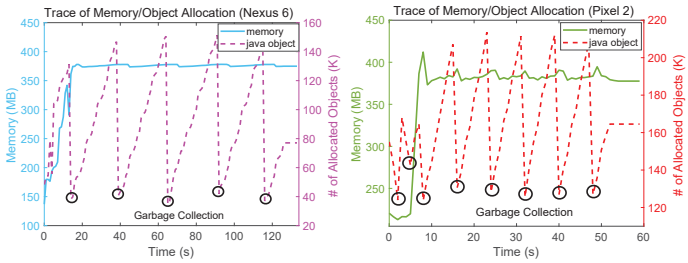
4.2.5 Robustness against Random Attacks. A *random attacker* tries to gain system access using his own walking data (gait) or data retrieved from a large database. Since behavioral patterns are extremely difficult to mimic by observation, we use Osaka as the database to launch attacks. These samples are entirely new to the model from unknown data distributions. We train users in the three datasets and enumerate through all the attacking samples (1684 spectrograms) for each user. As shown in Table 3, the success ratio is below 3%. Once the results are fused with 32 samples randomly selected from the training data, the ratio further declines to 1% in the worst case. This rate could be easily reduced to zero by incorporating high-level security mechanisms such as limiting the number of trials.

4.2.6 Inference on Mobile Devices. Fig. 6(a) shows time durations of making batched inference on mobile devices (from 4-56). Since less parallel resources are available on the mobile platform, the inference time increases almost linearly with the input batch size. The computation takes less than 1.5s for all the devices. Table 3 indicates that a batch of 32 samples is robust against random attacks. It takes less than 0.5s on Pixel2/Mate10 and 1s on Nexus 6/6P. If a single batch is not reliable, the system progresses to temporal decision fusion as



(a) (b)

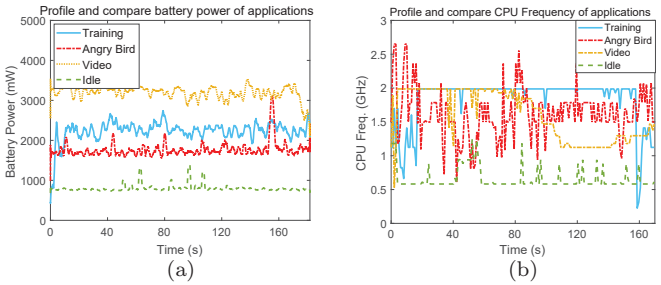
Figure 6: Inference on mobile (a) batched inference time on mobile; (b) process of decision fusion;



(a) (b)

Figure 7: Trace of memory/object allocation during mobile training (a) Nexus 6; (b) Pixel 2.

described in Section 3.2. Fig. 6(b) demonstrates the decision-making process. We set the false rejection/acceptance requirements to $\alpha = \beta = 0.01$. When the likelihood ratio hits the upper shaded area, the decision is to reject; otherwise, the decision is to accept. Normally, 5-6 batch iterations are needed to reach a confident decision. This takes about 6s and 1.5s on Nexus 6/6P and Pixel2/Mate10 respectively. To see the evolution, we select some hard samples and mix them with random samples. The classifier is less confident based on the single batch and it progresses to the next iteration until a shaded region is hit. The process can be thought as a competition between the decisions to either accept or reject. If a majority of the new data indicates positive, the decision is inclined to accept though a few false ones may drag the curve towards the opposite direction en-route. As we can see, decision fusion reduces prediction instability at a little cost of extended response time.



(a) (b)

Figure 8: Profiling battery power and CPU frequency of different applications

4.2.7 Profile System Overhead. Memory. We use the Android Profiler to measure the memory consumption of the app during training in Fig. 7. To save space, we show the traces of Nexus 6 and Pixel 2 (the oldest and newest of our collection). Nexus 6 has a quad-core of 4×2.7 GHz. Pixel 2 features an octa-core with 4×2.35 GHz plus 4×1.9 GHz CPUs. Once the app starts, it loads the native code, training samples and ConvNet model into the mobile memory. Sample paring is conducted on the device at the beginning. Since DL4J is not optimized for the mobile environment, the native/code occupies about 130 MB. When training is initiated, new objects are allocated and once the app approaches the assigned memory limit, a garbage collection is triggered to release the objects, which could pause the app for a minimum amount of time (several ms). When multi-threads are enabled in DL4J with OpenBLAS, the training process enjoys much better performance with an octa-core processor on Pixel 2. Hence, we see a steeper line of object allocation on Pixel2, which completes the training by only half of the time with Nexus 6.

Battery Power and CPU Frequency. We profile the battery power using the Monsoon power monitor[4] and CPU frequency by the Trepn Profiler[5]. We measure the battery power and average CPU frequency of the 4 cores on Nexus 6 while (1) training, (2) playing angry bird, (3) watching an MP4 video in MX player, and (4) idling, in Fig. 8. Training runs at 2.0 GHz set by the default governor and its battery power consumes at the level of 2000 mW, which consumes about 1% total battery during 2.5 mins. Training introduces an additional 28% energy overhead compared to angry bird, but consumes 25% less energy compared to watching a video. The results suggest that training consumes more energy than mobile games but less intensive than watching videos. Since model update is less time-sensitive compared to interactive apps, it can be delegated as a background service and scheduled on-demand while the phone is charging or idling. The default CPU governor can be also adjusted adaptively to optimize performance and power consumption.

5 CONCLUSION

This paper incorporates training on mobile devices and tackles the challenges from privacy, accuracy and performance. A comprehensive framework is designed to optimize training, inference to mitigate overfitting. The system is evaluated with a use case study of deep behavioral authentication and our extensive experiments demonstrate the security and robustness of the proposed design against intra-class variations and imposters that are out-of-distributions. We anticipate the presented system would offer insights and opportunities to enhance deep learning on mobile devices.

6 ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation under grant number CCF-1850045.

[4]Monsoon power monitor, https://www.msoon.com/
[5]Trepn Power Profiler, https://developer.qualcomm.com/software/trepn-power-profiler

REFERENCES

[1] M. A. Alsheikh, A. Selim, D. Niyato, L. Doyle, S. Lin, and H. Tan. 2016. Deep activity recognition models with triaxial accelerometers. In *AAAI Conference on Artificial Intelligence*.

[2] J. Ba and R. Caruana. 2014. Do Deep Nets Really Need to be Deep? In *Advances in Neural Information Processing Systems 27*.

[3] S. Chopra, R. Hadsell, and Y. LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 539–546 vol. 1.

[4] M. Gadaleta and R. Michele. 2018. IDNet: Smartphone-based gait recognition with convolutional neural networks. *Pattern Recognition* 74 (2018), 25–37.

[5] Google. 2019. *Low precision GEMM library.* https://github.com/google/gemmlowp

[6] S. Han, H. Mao, and W. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations* (2016).

[7] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (Nov 2012), 82–97.

[9] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[10] T. Hoang and D. Choi. 2014. Secure and privacy enhanced gait authentication on smart phone. *The Scientific World Journal* (2014).

[11] L. Jiang, R. Tan, X. Lou, and G. Lin. 2019. On Lightweight Privacy-preserving Collaborative Learning for Internet-of-things Objects. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI '19)*.

[12] X. Jiang, M. Kim, K. Lauter, and Y. Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1209–1222.

[13] Frank Jordan. 2019. *McGill Dataset.* https://www.cs.mcgill.ca/~jfrank8/data/gait-dataset.html

[14] H. Khan, A. Atwater, and U. Hengartner. 2014. Itus: An Implicit Authentication Framework for Android. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*.

[15] G. Koch, R. Zemel, and R. Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov 1998), 2278–2324.

[17] X. Li, Y. Zhang, I Marsic, A. Sarcevic, and R. Burd. 2016. Deep Learning for RFID-Based Activity Recognition. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems (SenSys '16)*.

[18] D. Liu, B. Dong, X. Gao, and H. Wang. 2015. Exploiting Eye Tracking for Smartphone Authentication. In *Applied Cryptography and Network Security*.

[19] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta. 2017. Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices. In *Proceedings of the 25th ACM International Conference on Multimedia (MM '17)*.

[20] M. Malekzadeh, R. G. Clegg, and H. Haddadi. 2018. Replacement AutoEncoder: A Privacy-Preserving Algorithm for Sensory Data Analysis. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*.

[21] A. Mathur, N. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar. 2017. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*.

[22] S. Mazilu, A. Calatroni, E. Gazit, A. Mirelman, J. M. Hausdorff, and G. Trster. 2015. Prediction of Freezing of Gait in Parkinson's From Physiological Wearables: An Exploratory Study. *IEEE Journal of Biomedical and Health Informatics* 19, 6 (Nov 2015), 1843–1854.

[23] P. Negi, P. Sharma, V. Jain, and B. Bahmani. 2018. K-means++ vs. Behavioral Biometrics: One Loop to Rule Them All. In *NDSS*.

[24] N. Neverova, C. Wolf, G. Lacey, L. Fridman, D. Chandra, B. Barbello, and G. Taylor. 2016. Learning Human Identity From Motion Patterns. *IEEE Access* 4 (2016), 1810–1820.

[25] T. Ngo, Y. Makihara, H. Nagahara, Y. Mukaigawa, and Y. Yagi. 2014. The Largest Inertial Sensor-based Gait Database and Performance Evaluation of Gait-based Personal Authentication. *Pattern Recogn.* 47, 1 (Jan. 2014), 228–237.

[26] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu. 2018. DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[27] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang. 2015. User Verification Leveraging Gait Recognition for Smartphone Enabled Mobile Healthcare Systems. *IEEE Transactions on Mobile Computing* 14, 9 (Sep. 2015), 1961–1974.

[28] J. Roth, X. Liu, and D. Metaxas. 2014. On Continuous User Authentication via Typing Behavior. *IEEE Transactions on Image Processing* 23, 10 (Oct 2014), 4611–4624.

[29] R. Saa, J. Milica, M. Nadja, and K. Vladimir. 2014. Gait characteristics in patients with major depression performing cognitive and motor tasks while walking. *Psychiatry Research* 217, 1 (2014), 39–46.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[31] T. Shruti, G. Karan, S. Shweta, B. Ranjita, and R. Ramachandran. 2018. Privado: Practical and Secure DNN Inference. *ArXiv* abs/1810.00602 (2018).

[32] K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[33] A. Wald. 1945. Sequential tests of statistical hypotheses. *The annals of mathematical statistics* 16, 2 (1945), 117–186.

[34] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. 2016. A Discriminative Feature Learning Approach for Deep Face Recognition. In *Computer Vision – ECCV 2016*.

[35] M. Xu, J. Liu, Y. Liu, X. Lin, Y. Liu, and X. Liu. 2019. A First Look at Deep Learning Apps on Smartphones. In *The World Wide Web Conference (WWW '19)*.

[36] W. Xu, G. Lan, Q. Lin, S. Khalifa, M. Hassan, N. Bergmann, and W. Hu. 2019. KEH-Gait: Using Kinetic Energy Harvesting for Gait-based User Authentication Systems. *IEEE Transactions on Mobile Computing* 18, 1 (Jan 2019), 139–152.

[37] X. Zeng, K. Cao, and M. Zhang. 2017. MobileDeepPill: A Small-Footprint Mobile Deep Learning System for Recognizing Unconstrained Pill Images. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*.

[38] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. Phuong. 2018. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning.. In *IJCAI*. 3933–3939.

[39] Y. Zhang, G. Pan, K. Jia, M. Lu, Y. Wang, and Z. Wu. 2015. Accelerometer-Based Gait Recognition by Sparse Representation of Signature Points With Clusters. *IEEE Transactions on Cybernetics* 45, 9 (Sep. 2015), 1864–1875.

[40] N. Zheng, K. Bai, H. Huang, and H. Wang. 2014. You Are How You Touch: User Verification on Smartphones via Tapping Behaviors. In *2014 IEEE 22nd International Conference on Network Protocols*.