Explore Training of Deep Convolutional Neural Networks on Battery-powered Mobile Devices: Design and Application

Cong Wang, Yanru Xiao, Xing Gao, Li Li, Jun Wang

Abstract—The fast-growing smart applications on mobile devices leverage pre-trained deep learning models for inference. However, the models are usually not updated thereafter. This leaves a big gap to adapt the new data distributions. In this paper, we take a step further to incorporate training deep neural networks on battery-powered mobile devices. We identify several challenges from performance and privacy that hinder effective learning in a dynamic mobile environment. We re-formulate the problem as metric learning to tackle overfitting and enlarge sample space via data paring under the memory constraints. We also make the scheme robust against side-channel attacks and run-time fluctuations. A case study based on deep behavioral authentication is conducted. The experiments demonstrate accuracy over 95% on three public datasets, a sheer 15% gain from multi-class classification with less data and robustness against brute-force and side-channel attacks with 99% and 90% success, respectively. We show the feasibility of training with mobile CPUs, where training 100 epochs takes less than 10 mins and can be boosted 3-5 times with feature transfer. Finally, we profile memory, energy and computational overhead. Our results indicate that training consumes lower energy than watching videos and slightly higher energy than playing games.

Index Terms—On-device machine learning, privacy preservation, deep metric learning, behavioral authentication

1 Introduction

Propelled by the latest advance in mobile processors, smartphone becomes an ideal platform to conduct deep learning tasks at the data source. Without offloading private data to the cloud and consuming data bandwidth, users enjoy more coherent interaction and better experience. This trend quickly reignites innovations in the saturated mobile industry as new applications quickly emerge from photo classification, beauty enhancement, text prediction, speech recognition and movement tracking [1]. Since deep neural networks are compute-intensive, the recent efforts mainly focus on improving computation and memory efficiency by quantization [2], model compression/sparsification [3]–[5] and distillation [6], [7]. These techniques can be applied during model deployment to reduce run-time redundancies on mobile devices.

Most of the existing applications and optimizations focus on *inference* from a pre-trained model, whereas leave *training* on battery-powered mobile devices largely unexplored. Nevertheless, being able to infer from a static model still has a significant gap from being cognizant, since machine learning relies on the assumptions that the test samples are independently and identically drawn from the same distribution of training. Deep classifiers are not good at extrapolation when data comes from a different distribution, but it is quite common in mobile applications. Take the ongoing research of behaviorial authentication [8]–[11] and activity recognition [12], [13] for example, behaviorial patterns may evolve due to sickness, injury and emotion, thus intensify the intra-class variations and hamper classification. This requires the model to adapt to the new distribution, where constant training and finetuning are expected. Therefore, being truly intelligent should bring *training* back into the loop.

Due to resource consumption, a natural solution is to securely offload training to the cloud. For example, one can host an enclave for each client in the cloud [14], securely aggregate training data from users, keep all the models updated and provide downlink accessibility on-demand; or use homomorphic encryptions to obfuscate both private data and model, while the cloud can still conduct meaningful computations on encrypted data or model weights [15]–[17]. Other options include pre-processing to (randomly) project user data into non-sensitive representations [18], which is yet to offer a rigorous security guarantee and often comes with an accuracy loss.

Can we incorporate training on mobile devices as well? In addition to the foreseeable challenges of computation and power, what are the gaps between mobile environments and machine learning that limit deep learning algorithms to perform as they should? Previous efforts have studied deep learning on mobile platforms. However, these platforms have no essential difference from the cloud setting with proprietary GPU and external power. By contrast, in this paper, we consider *battery-powered* smartphones running Android that are thermally and electrically limited.

Mobile environments have fundamental difference from the cloud in terms of *labeled data*, where labeling is usually crowdsourced offline. Despite the abundance of mobile data, interactive labeling from the user degrades useability,

- *C. Wang and Y. Xiao are with the Department of Computer Science, Old Dominion University, Norfolk, VA, USA, E-mail: {c1wang, yxiao002}@odu.edu*
- *X. Gao is with the Department of Computer Science, University of Memphis, TN, USA, Email: xgao1@memphis.edu*
- *L. Li is with Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China, Email: li.li@siat.ac.cn*
- *J. Wang is with Futurewei Technologies, Santa Clara, CA, Email: jun.wang2@huawei.com*
- *Correspondence to c1wang@odu.edu.*

whereas having a small dataset would lead to overfitting and low accuracy. Second, since the read access to many sensors is not restrictive [22]–[25], learning should take extra precautions to these side channels, especially in security-critical applications. Third, mobile data is inherently noisy; decisions based on a one shot of inference is not reliable. Multiple inferences should be fused across the spatial-temporal domain for a more confident decision. Finally, as training typically takes hundreds of epoches using back-propagation, it requires to speed up the convergence in a privacy-preserved manner.

This paper takes a first step to tackle this multi-faceted challenge from *accuracy* and *privacy*. It is realized through a comprehensive use case study of deep behavioral authentication, a promising, second-factor authentication that does not require deliberate attention from users. For *accuracy*, we re-formulate the classification problem in a different way by making samples into pairs under the device's memory constraint, and learn a deep metric to mitigate overfitting due to data scarcity [26], [27]. Then we 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 of learning. For *privacy*, we develop a defense mechanism to reject fraudulent samples sniffed from side channels. It is achieved by embedding a noise fingerprint inside the sensing signal and supervising the neural network to distinguish between genuine samples from the ones with noise perturbation. We also implement feature transfer to speed up training convergence on mobile, while securing all the intermediate activations/model parameters. The main contributions are summarized below.

- We conduct both training and inference on battery-powered mobile devices to preserve privacy and learn effectively in a dynamic mobile environment. The implementation on Android demonstrates that training is not only feasible but also quite fast with feature transfer (within 5s/epoch on Huawei Mate10 for 400 samples).
- We tackle the overfitting problem due to the lack of labeled data by paring samples under the memory constraint and learn a deep metric to enhance the discriminative power of the model. In our case study, we analyze and evaluate various types of data representations and loss objectives thoroughly. Our experiments demonstrate 10-15% improvement of authentication accuracy on different datasets and achieve an accuracy of 0.94 on a large dataset with 153 participants.
- We successfully mitigate potential side-channel leaks and reduce attack success ratio below 10% while preserving usability of benign applications.
- We evaluate the framework extensively on public datasets and profile learning performance and 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 studies related literatures. Section 3 and Section 4 present the system model and design. Section 6 evaluates the framework through a case study and Section 7 concludes this work.

2 Related Works

2.1 Privacy Preservation

Privacy preservation has been extensively studied recently in the machine learning community. The research mainly focuses on three different directions. The first is the algorithmic direction such as differential privacy [28] and data projection [18]. Differential privacy introduces noise into the training process so adversaries cannot detect the presence or absence of a user [28]. Autoencoder is utilized to transform sensitive features into a latent space for non-sensitive inference [18]. Since these approaches have to make a balance between useability and privacy, it is hard to provide rigorous security guarantees and there is always an associated cost of accuracy loss. The second direction is homomorphic encryption that allows curious third parties to perform meaningful computations on encrypted data [15], [16]. CryptoNets use fully homomorphic encryption to encrypt the data from the client and receive an encrypted prediction from the cloud. A square activation function is adopted to bridge the gap between the cryptographical and neural operations, which is only suitable for inference computation. The work of [15] designs a two-party protocol to protect both the data and the model using a partially homomorphic encryption. Since homomorphic encryption and decryption are computationally intensive, the computation, energy and network overhead would be prohibitive for mobile devices.

From the system perspective, the third approach is to implement computations directly at the data source, so private information never leaves the device. The existing works either develop new variants of applications such as activity recognition [12], [13] and mobile vision [29], [30], or explore the design space to optimize the performance of conducting inference on mobile devices [2]. These techniques aim to reduce the run-time redundancy of the neural networks. A direct method is to round the original model parameters in 32-bit floating point to 8-bit integer, so 75% size can be saved [2]. Another approach is to prune the connections with near-zero weights either after training [4] or during training [5] for a sparse network. A large and complex teacher network can be also used to train a small student network for comparable results, thus distilling the knowledge to run the small network on mobile devices [6], [7]. These methods are effective for running inference on mobile devices. Nevertheless, they left training out-of-the-loop, especially in the applications [12], [13], [29], [30]. This leaves the model without a continuous, timely engagement to the dynamics in mobile applications. This work fills such gap by exploring the challenges of training on mobile devices from an application perspective.

2.2 Behavioral Authentication

Smartphone features a variety of sensors to capture behavioral information using acceleration, gyroscope, etc. Behavioral biometrics such as gait [8], [9], keystroke dynamics [10] and eye movement [11] 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 [31], 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 [8], whereas data outliers, abrupt changes could easily mislead these techniques. Deep neural networks are used in [32] to train a homogeneous model for recognition. Yet, they do not address issues of where and how the model should be trained and the potential threats from side channel leaks when the deep learning algorithms are implemented on mobile devices.

3 System Architecture and Model

In this section, we describe the system architecture and threat model for the deep behavioral authentication, whereas the general framework shares similarities with other sensing applications such as activity recognition. The proposed system is depicted in Fig. 1.

Threat Model. The authentication module reads authentic sensor data from the standard API (e.g., `SensorManager` in Android). We assume that sensor data is trustworthy as its integrity can be protected by hardware security extension technologies such as ARM TrustZone [34]. Meanwhile, sensor data is available to all applications, including malware, since modern OS like Android or iOS does not set restrictions to accessing sensors. By exploiting sensor data (e.g., accelerometer, gyroscope, magnetometer), numerous side-channel attacks [22]–[25] have been reported to sniff PIN/swipe patterns, password, app usage or even speech. We assume that attackers attempt to break the biometric authentication by utilizing sensor data.

The main performance metric is the authentication accuracy, measured by whether the system can defend against spoofing attacks and recognize its owner. Specifically, we consider two types of attackers: *passive* and *active*. *Passive* attackers [10] 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, the passive attacker can only retrieve data from a large public database and launch brute-force attacks to unlock the device. *Active* attackers can directly and stealthily collect sensory data from the user's smartphone. This could be achieved by tricking the user into installing a third-party app. We further assume the powerful active attacker can generate the exact same gait pattern as the sniffed sensory data by programming an apparatus such as a robot [35].

One effective countermeasure to side-channel attacks is sensor data obfuscation [24], which injects random noise to user-level apps so that malware cannot recognize sensitive operations. The noise level should balance the usability of benign apps while obfuscating malicious ones. Thus, to mitigate side-channel attacks, the obfuscation technique is employed in the target smartphone by wrapping the `SensorManager` API. For instance, Slogger [36] can inject noise into various sensor outputs. The noise is transparent to the authentication module, which is integrated in the mobile operating system. While applications (including malware) can only obtain obfuscated data through the wrapped interface, they can apply various denoising techniques to restore the original data.

Finally, the communication between the device and the cloud is secured by SSL. Since our design is centered around on-device implementation, the cloud only plays a secondary role to provide samples from the negative classes. We assume cloud providers are honest but curious: they follow protocols but are free to use what they see to learn private information from users.

4 Framework Design

This section presents the main design of the framework, including the learning technique, decision-making mechanism and opportunities to speed up the computation.

4.1 Deep Metric Learning

Although the smartphone accumulates considerable sensing data at runtime, labeling requires external efforts from the user. For example, the authentication application would ask the user to record a segment of behavioral data during the bootstrapping phase and use these as the ground truth for training. Existing approaches of classification typically use the *softmax* loss to output a probability for each predicted class. With a total of k classes and n samples in each class, it learns from the $\mathcal{O}(kn)$ samples. For each class, the softmax function only examines the n samples. When n is small, the model might be subject to serious overfitting and perform poorly during the test time.

A solution is to make samples into pairs so each sample is paired with the rest $\mathcal{O}(kn)$ samples, and a similarity distance metric can be learned using the *Siamese Network* [26], [27]. This way, the input is expanded by a factor of kn and the discriminative power is enhanced. As shown in Fig. 1, the Siamese Network incorporates two branches of identical convolutional neural networks 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.

Joint Loss. A slightly different loss function $\mathcal{L}_s$ is proposed in [27], that maps dissimilarity into a probability prediction with sigmoid activation so the network can be

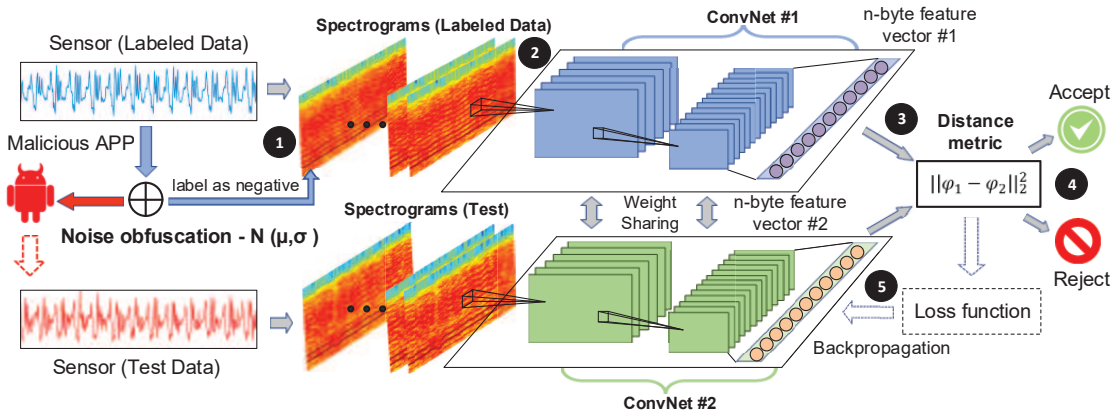


Fig. 1: System architecture on mobile devices: ❶ it takes raw sensor inputs, transforms them into mid-level representations (spectrograms [33]); ❷ processes the representations with the neural network; ❸ computes a distance metric from the feature vectors; ❹ generates a decision; ❺ backpropagates the error if training is scheduled.

trained with cross-entropy loss. The advantage is that no margin needs to be pre-determined,

$$\mathcal{L}_s = \sum\nolimits_{i,j\in\mathcal{D}} y(\varphi_1^{(i)}, \varphi_2^{(j)}) \log p(\varphi_1^{(i)}, \varphi_2^{(j)}) + (1 - y(\varphi_1^{(i)}, \varphi_2^{(j)})) \log(1 - p(\varphi_1^{(i)}, \varphi_2^{(j)})). \quad (2)$$

The construction of the loss function plays an important role to capture similarity in different applications. To capitalize from the potential advantages of distance and probabilistic metrics, we combine them into a new joint loss function. The goal is to minimize the total loss $\mathcal{L}_t$ with a balancing parameter α,

$$\mathcal{L}_t = \mathcal{L}_c + \alpha \mathcal{L}_s, \quad (3)$$

where $\mathcal{L}_s$ is the cross-entropy loss in Eq. (3) and $\mathcal{L}_c$ is the contrastive loss in Eq. (1). α is a scalar used for balancing the two functions. We take l_2-normalization of distance $f(\varphi_1, \varphi_2) = \|\varphi_1 - \varphi_2\|_2$ and evaluate these different formulations in Section 6.

4.2 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 based on [37]. 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

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 𝒩, R = r^2, |𝒩| = n_s rs.
for T ← 1, ··· , R do
    ℛ ← ℛ + (i ∈ 𝒩).
for T ← R + 1, ··· , n_s rs do
    if probability p > R/T then
        ℛ ← ℛ − (i ∈ ℛ) + (i ∈ 𝒩).
```

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.

4.3 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) [38]. SPRT progresses by assessing a likelihood ratio λ_n for the n-th observation,

$$\lambda_n = \frac{p(y_1, \cdots, y_n|H_1)}{p(y_1, \cdots, y_n|H_0)} = \prod_{i=1}^{n} \frac{p(y_i|H_1)}{p(y_i|H_0)}. \quad (4)$$

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,

$$p(d_i|\mu, \sigma^2) = 1 - \phi(\frac{d_i - \mu}{\sigma^2}), \quad (5)$$

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 (4) and (5), the ratio is,

$$\frac{p(y_i = 0|H_1)}{p(y_i = 0|H_0)} = \phi(\frac{d_i - \mu}{\sigma^2})/(1 - \phi(\frac{d_i - \mu}{\sigma^2})) \quad (6)$$

$$\frac{p(y_i = 1|H_1)}{p(y_i = 1|H_0)} = (1 - \phi(\frac{d_i - \mu}{\sigma^2}))/\phi(\frac{d_i - \mu}{\sigma^2}) \quad (7)$$

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

$$S_n^* = \begin{cases} H_0, \lambda_n \leq B \\ H_1, \lambda_n \geq A \\ continue, B < \lambda_n < A \end{cases} \quad (8)$$

We set the two thresholds A and B suggested by [38], $A = (1 - \beta)/\alpha$, $B = \beta/(1 - \alpha)$. 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 and 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 deviations thereby closing the loop of learning on mobile devices. The scheme is summarized in Algorithm 2 and evaluated in Section 6.8.

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 $\mathcal{D}_t$.

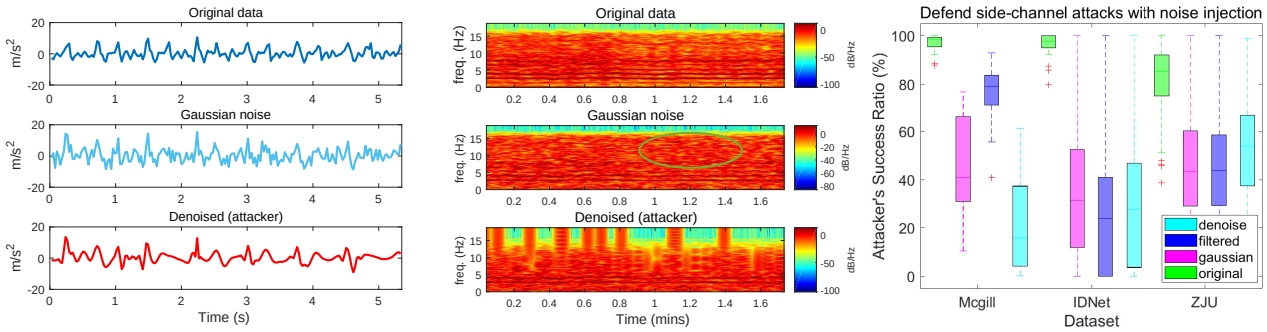
4.4 Defend Side-channel Leaks and Active Attacks

Although well-trained neural networks could achieve high accuracy, the sensitive data might be also leaked via side channels. Since acquiring data from the motion sensors does not require any permissions, curious third parties could exploit these data to infer health or mental condition such as predicting Parkinson's disease, depression from gait patterns [46], [47]. Further, skilled attackers can trick the user into downloading an app that stealthily captures the motion data, and then replay them to gain the access via programming an apparatus [35].

A typical countermeasure is to obfuscate the sensor output by injecting random noise [24]. However, our experiments show that simply injecting random noise into the sensing signal still fails to fully prevent the attack. Specifically, we obfuscate the data with a *zero-mean* gaussian noise, whose standard deviation is set to equal the original signal over a moving window. As circled in the middle picture of Fig. 2(b), this obfuscating operation introduces a few new energy components at higher frequencies in the spectrograms, which enable the authentication module to recognize attackers to some extent. Fig. 2(c) shows the attacker's success ratio. Without any protection, the attacker can easily accomplish 80-100% success rate (the true positive rate). With noise injected, the success rate drops to an average of 50%. However, this success rate is still not sufficiently secure. In extreme cases, some individuals are still subject to 100% attack success rate even when noise is injected.

One reason that simple noise injection does not work here is that neural network is robust to random noise. It extracts a meaningful combination of features towards a minimization of the loss objective, and serves as an information bottleneck that finds a compressed mapping of input that preserves maximally possible information of the output [39]. Thus, redundant information including small noise and interference, which does not interfere with the main structure, is thrown away. As a result, the attackers can still succeed. Obviously, a successful obfuscation requires raising the standard deviation of the noise (e.g. surpassing the std of the signals) to generate larger noise. However, it will inevitably impact the usability of legitimate apps.

Furthermore, attackers can apply various denoise techniques to potentially raise the success rate if the original waveform is not changed by the denoise method. In our preliminary experiment, we measure the attackers' success rate by applying two denoise methods: total variation proximity operators [40] and 1D gaussian filter. As shown in the spectrograms in Fig. 2(b), the new energy components at higher frequencies are removed by denoising. In our



(a) (b) (c)

Fig. 2: Preliminary assessment of simple noise injection (a) raw sensing data with/without noise; (b) relevant spectrograms; (c) attacker's success ratio.

experiment, by applying appropriate denoising methods on Mcgill and ZJU datasets, attackers can achieve higher success rates.

Our approach. We propose a new defense mechanism against strong adversary by making a small extension, that supervises the neural network to learn the injected noise and use it as a hidden fingerprint. In particular, our approach develops on the fundamental that the power of neural networks can even fit unstructured random noise with random labels [41]. If the pair of genuine and noised data (genuine plus noise) is labeled as negative, we are supervising the Siamese Network to map them into different areas in the feature space. Then, the noise data becomes a *hard example* [42] that looks similar to the genuine one. With the superb memorization capability of the neural network, it is also able to fit the noise part in brute force. By labeling them as negative, the system forcibly learns their nuances, whereas the attacker can only sniff/use the noise data and get rejected by the system. While the naive way of noise injection requires a large noise level for successful obfuscation, boosted by the unique feature of neural network, our approach can significantly reduce the required level of noise without sacrificing much usability.

To mitigate the impact of possible denoising from the attacker, the system can further predict the potential classical denoise algorithms that might be used by attackers. The authentication module can generate the denoised pairs beforehand and similarly label them as negative for training. The detailed evaluation of our proposed defending system is presented in Section 6.7.

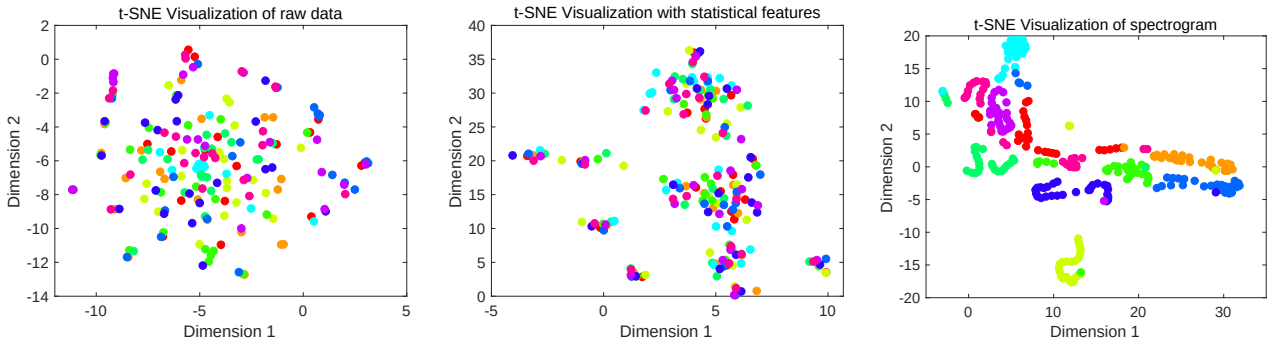
4.5 Speed up Convergence via Privacy-Preserved Feature Transfer

Training learns hundreds of thousands of parameters through backpropagation, which could take more than hundreds of epochs till convergence. The previous work proposed to partition the neural network between the cloud and mobile device in a layer-wise manner [21]. We build on this approach to speed up convergence via feature transfer. With domain similarities, knowledge learned from the source can be efficiently repurposed for the target domains. For neural networks, the high-level features learned from the first few layers are more generic, while the low-level features are more specific to the classification tasks [43] (e.g., the early layers learn general features like edge detectors to identify the concentration of frequency energy from the sensing signals).

To initiate, the cloud (*source*) and the mobile (*target*) agree on a partial network structure of the first k layers, e.g., the first two convolutional layers. The agreement includes the model weights and hyper-parameters. For the target model, a few adaptation layers are introduced. Thus, the high-level features could be efficiently reused on user's mobile device. We can utilize a source model $\mathcal{M}_s$ trained on the public dataset with n_s classes in the cloud, and transfer the learned features for the n_t classes in the target model $\mathcal{M}_t$, when the source and target domains do not overlap.

Specifically, samples x from the source domain are passed through $\mathcal{M}_s$ until the k-th cutoff layer, where x is represented as an n-byte feature vector. Next, the model parameters $\mathcal{M}_s$ are transferred to the target model $\mathcal{M}_t$ for the first k layers, along with all the feature vectors (transmitted via a secure channel - SSL). To initiate training on mobile, the target model freezes weights of the first k layers. The error is backpropagated from the last layer to the $(k+1)$-th layer. The weights of the adaptation layers are adjusted according to stochastic gradient descent. Note that the cloud is not aware of the layer structure or model weights beyond the k-th layer on the mobile devices. We show by experiments that this approach has great potentials of adaptation, such as allowing the source and target to have different loss functions (softmax for $\mathcal{M}_s$ and contrastive loss for $\mathcal{M}_t$) and heterogenous sensor hardware with different sampling frequencies. The target model can still learn effectively and enjoy massive speed-up of convergence with little accuracy loss.

Remarks on privacy: Other than the active attackers, privacy exploits attempt to reconstruct the original data from feature activations [44] or model parameters [45]. Our design is robust against these exploits since: 1) activations generated from the target model are kept on mobile thus curious cloud providers cannot invert the private data; 2) though data can be recovered from the shared weights of k layers on mobile by curious users, the data is public and carries little business value; 3) the model weights beyond the k-th layers on mobile are not disclosed to anyone else, hence a third party cannot recover private data from the model parameters.



(a) (b) (c)

Fig. 3: t-SNE visualization (best view in color; each color represents samples from an individual) (a) raw walking data (b) nine statistical features (c) spectrogram.

5 Implementation

5.1 Data Pre-processing

Data processing lays the foundation to achieve high accuracy. Here, we formally discuss the intuition behind using spectrograms to represent sensing signals gathered by accelerometer sensors. Unlike images, the accelerometer signal is one-dimensional time series. Existing research mainly works in the time-domain and requires cycle extraction [32] or segmentation [25]. Cycle extraction looks for cyclic patterns between local minima or maxima algorithmically, but is prone to error in the presence of noise. Segmentation divides the signal into many overlapped pieces that expands the dataset by many folds. Here, we adopt a new approach to model walking data as speech and demonstrate its advantages in the following.

Walking consists of a set of motions from the body parts (torso and limb), which shares similarities with speech from their generation mechanisms. While speaking, the pulse from vocal cords is modulated in frequency through the throat cavity and reshaped by the articulators (tongue, mouth, lips) to produce sound. Gait signals generate a similar pattern from the body parts. Based on these observations, it is reasonable to model gait as speech, which is typically analyzed in spectrogram [33].

A spectrogram uses three dimensions to represent signal energy as a function of time (x-axis) and frequency (y-axis). It breaks data into segments of short intervals, takes short-time Fourier transform in each segment and assigns frequency spectrums into different bins of magnitude. Each bin stands for the frequency scale perceived. Spectrogram concatenates multiple quasi-stationary cycles to generate a 2D output. This way, learning can be performed effectively using convolutional neural networks.

A compelling advantage of spectrogram is suggested by Fig. 3 (visualized by the t-SNE tool to reduce the high-dimensional data into 2D [48]). Fig. 3(a) shows the raw sensing signal and Fig. 3(b) visualizes the data with nine statistical features. Though these statistical features form a distinguishable trend, they are still not powerful enough; in sharp contrast, data points are clustered in a more organized manner in Fig.3(c), so it is much easier to build a classifier and recognize different individuals.

5.2 Model Development

Model architecture determines learning capability, memory requirement, and computational intensity. In this paper, we evaluate three convolutional neural network architectures extended the families of *LeNet* [49], *VGG* [50] and *MobileNetv2* [3]. Though an alternative is to use the recurrent neural networks, their computation intensity are much higher on the mobile devices. We customize these classic models to add or prune layers in order to yield similar input dimension at the dense layer as their original implementation with the ImageNet. The spectrograms of (x, y, z) axis are stacked vertically to form 33×42 images.

In particular, LeNet repeats two blocks of 5×5 convolutional and max pooling layers followed by densely connected layers. We add one more 3×3 convolutional layer and prune one dense layer to get 4 weight layers (therefore the name LeNet4). VGG repeats two 3×3 convolutional layers to achieve similar receptive field with the 5×5 convolution, but much less computation/parameters. Multiple such blocks are stacked to learn complex relations among the features. We repeat the blocks three times and introduce one more convolutional layer before the last max pooling, thus making VGG8 a heavy-weight network with 8 weight layers. We also implement the latest MobileNetv2. The model stacks inverted residual blocks (inv_res_bl) to take low-dimensional representation, and then expands to the high-dimension for efficient feature extraction by the depthwise convolution. The blocks are connected with bypass links to make deeper structures trainable with less degradations. The max pooling layer is replaced by a convolution stride of 2, e.g. $(1, 1, 2)$ represents two blocks with a stride of 1 followed by a block with a stride of 2. Table 1 summarizes the model architectures and layerwise parameters.

5.3 Mobile Development

The choice of the software framework is crucial since training requires backpropagation. Despite a handful of available frameworks, most of them (e.g. *Tensorflow Lite* [51]–[53]) have tailored backpropagation and left only the inference part to compute from pre-trained models. This way, no intermediate gradient values need to be stored and the memory/code can be optimized. In this paper, to enable training on the mobile device, we develop the system on a Java-based framework called *DL4J* [54]. Since the two Siamese branches

are identical, only one copy of the model is stored in memory. 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.

6 Use Case Study of Behavioral Authentication

This section conducts a thorough evaluation based on the deep behavioral authentication. The main goals of the evaluations 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 both random and active attacks; 4) profile performance and overhead on various smartphone models.

6.1 Dataset and Experimental Settings

To make the benchmarks comparable, the experiments are based on public datasets: Mcgill [55], IDNet [56], ZJU [57] and Osaka [58] gait datasets. Note that this paper focuses on algorithm design and system integration rather than collecting, analyzing or deriving data from human subjects. Thus, an IRB approval is not required. 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 on two different days. IDNet is collected in a more vibrant environment with different types of phones and dresses from 50 people. ZJU collects gait data from 153 individuals in 3 different sessions using 5 body sensors of low sampling rates. Osaka records 1-minute walk of 744 subjects. Due to short recordings (only 1-2 spectrograms), we cannot perform meaningful training so it is utilized as a large database from which attackers may launch random attacks. Since some individuals have much less or missing data in IDNet and ZJU, we remove those individuals for data balance. This ultimately brings them to 30 and 136 individuals respectively.

The datasets are split into 80% for training and 20% for testing. For the siamese network, the training 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. 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 also evaluate the trade-offs between false rejection (the genuine user is falsely rejected) and false acceptance (an imposter is falsely accepted) using different margin threshold. We set the margin $m = 1.5$ in the contrastive loss (Eq. (1)) and $\alpha = 0.1$ in the joint loss (Eq. (3)). For fast prototyping, we first develop the model and evaluate authentication accuracy, security and performance in *Tensorflow* [51] with Nvidia Tesla P100 GPU, and then develop the learning module on Nexus 6/6P, Huawei Mate 10 and Google Pixel2 using DL4J [54]. A large batch size of 128 is used while training on GPU. 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.

6.2 Authentication Accuracy

We first evaluate the authentication accuracy by comparing models, data representation and learning mechanisms on different datasets in Table 2. We validate the choice of spectrogram by comparing with the pre-processing technique of sliding window (SW) [25] on the temporal data both using softmax (cols. 1, 2). As envisioned by the t-SNE visualization of Fig.4, spectrogram achieves a significant accuracy gain of over 10% (col. 4). A one-class SVM (osvm) is used in [32] to detect outliers from imposters. It takes features from the last convolutional layer learned from the softmax function to train an osvm using positive samples only. Unfortunately, though osvm can handle 80-90% outliers, it fails to generalize to the positive samples, which results in high rate of false rejections. Thus, the total accuracy is just slightly better than random guesses (col. 3).

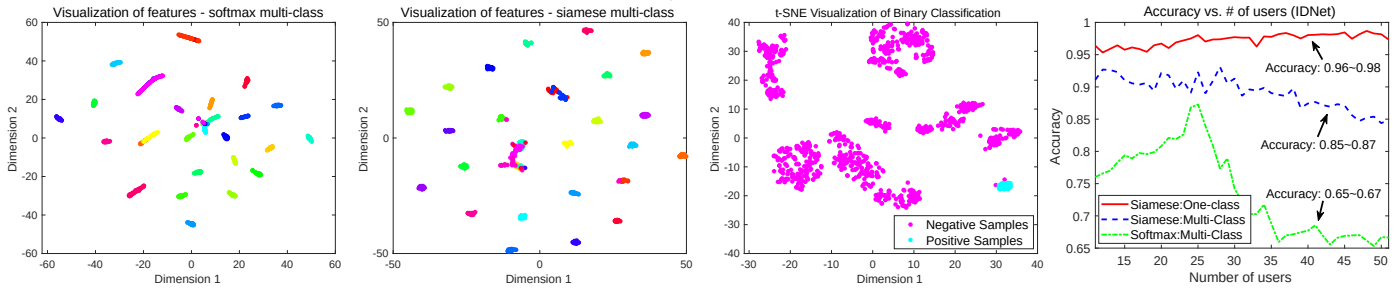
Softmax vs. Contrastive Loss. Our motivation to use the Siamese Network is because of the higher discriminative power on small data. To validate, we first visualize the features learned by softmax and siamese (contrastive loss) in Figs. 4(a) and (b), where the colors represent the feature vectors of 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 [59] (softmax tends to underperform). Contrastive loss from the siamese network offers improvements by mapping feature activations into a condensed, compact set of spaces. This validates the higher discriminative power of deep metric learning than softmax especially with less training data. Not only via feature visualization, the authentication accuracy also indicates 8-15% improvements between the two methods (col. 2 and 4-6 of Table 2).

Binary vs. Multi-class Classification. We discuss the impact of formulating the problem into either the binary or multi-class classification problem. Multi-class classification requires all the pairs between different classes to be labeled whereas binary only labels one vs. the rest. The former is more suitable for recognition tasks where a centralized model is trained to identify different users. The recognition model can be also migrated to the mobile devices for authentication [32]. However, it is subject to potential security risks when a malicious end user attempts to invert training data of other individuals [45]. In addition, we investigate the performance gap between the two formulations in terms of model accuracy.

To simulate limited mobile storage, only 20% data from the training set is used for binary classification but evaluated on the entire test set. This is challenging for recognition since the neural network can only "see" from a small subset of training data. A model is trained for each individual and the results are averaged. Fig. 4(c) visualizes binary classification. It only distinguishes the positive samples from the rest and the negative samples can be mapped to similar locations in space without causing an error. Nevertheless, multi-class classification still has to separate all the individuals by a margin, which makes it difficult to differentiate hard samples (Figs. 4(a)(b)). To test the scalability of multi-class

LeNet4		VGG8		MobileNetv2	
#layer blocks	# param.	# layer blocks	# param.	# layer blocks	# param.
$32\times$conv2d$(5,5)$+pool	0.96K	$2\times(64\times$conv2d$(3,3)$+pool$)$	39.2K	conv2d$(3,3,2)$	1.9K
$64\times$conv2d$(5,5)$+pool	51.5K	$2\times(128\times$conv2d$(3,3)$+pool$)$	222.5K	$(16,32)\times$inv_res_bl$(1,1,2)$	47.3K
$32\times$conv2d$(3,3)$	51.4K	$3\times(128\times$conv2d$(3,3)$+maxpool$)$	444.3K	conv2d$(1,1)$	1.1K
dense(128)	82.5K	dense(128)	327.8K	dense(64)	61.8K
contrastive/x-entropy loss	186.36 K	contrastive/x-entropy loss	1033.8K	contrastive/x-entropy loss	112.1 K

TABLE 1: Summary of model architectures



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

Fig. 4: Multi-class and binary classification via t-SNE visualizations (a) softmax (multi-class); (b) siamese (multi-class); (c) siamese (binary); (d) accuracy vs. user number. (Best view in color)

		baseline			siamese multi-class			siamese binary-class (20% data)		
		softmax(sw)	softmax(spgm)	osvm	contrastive	x-entropy	joint	contrastive	x-entropy	joint
Mcgill	LeNet4	0.774	0.881	0.542	0.918	0.940	0.925	0.966	0.934	**0.975**
	VGG8	0.752	0.902	0.672	0.925	0.952	0.931	0.962	0.906	**0.973**
	Mobilenet	0.682	0.811	0.581	0.865	0.926	0.923	0.847	0.901	**0.957**
IDNet	LeNet4	0.726	0.842	0.552	0.884	0.903	0.910	0.937	0.899	**0.943**
	VGG8	0.764	0.875	0.561	0.916	0.934	0.915	0.908	0.901	**0.941**
	Mobilenetv2	0.770	0.776	0.591	0.876	0.912	0.867	0.910	0.921	**0.945**
ZJU	LeNet4	0.442	0.646	0.511	0.681	0.804	0.779	0.941	0.926	**0.972**
	VGG8	0.463	0.743	0.523	0.769	0.841	0.800	0.936	0.851	**0.981**
	Mobilenetv2	0.591	0.471	0.510	0.706	0.778	0.743	0.895	0.835	**0.921**

TABLE 2: Model accuracy of different loss functions for the siamese network

classification, we show the results in Fig. 4(d) by increasing the number of classes in IDNet. The accuracy declines with a growing number of classes in the system. Hence, model capacity should keep growing as new users subscribe to the service. This would require extensive maintenance efforts in distributed mobile environments. As projected in Fig. 4(d), accuracy is independent from the system scale using binary classification with a fixed network architecture.

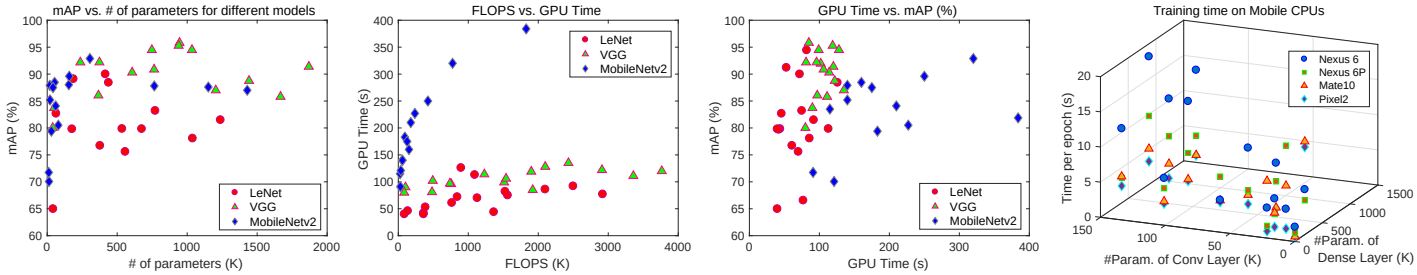
Table 2 summarizes the overall accuracy comparison. With multi-class classification using the Siamese Network, accuracy still declines a little with an increasing number of classes (e.g. from 0.952 of Mcgill with 20 people down to 0.841 of ZJU with 136 people). By reducing the problem into binary classification, the accuracy stays above 90%. Among them, the new joint loss accomplishes the best accuracy with over 95% correctness. This is because the joint loss balances the two loss functions and combines the model outputs for higher fidelity.

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 would further investigate this issue in our future work. Finally, we 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. This indicates that networks particularly optimized on model parameters and computer vision tasks may perform poorly on mobile sensing tasks, compared to simple solutions of stacking convolutional layers such as VGG8.

6.3 Resource Requirement

To quantify the performance and resource requirements of the mobile sensing task, we conduct more experiments to illustrate the relations between model parameters, floating point operations (FLOPS), and accuracy in Fig. 5. 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. 5(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. 5(b) also indicates that it incurs nontrivial GPU time if the FLOPS increase. Fig. 5(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. 5(d) shows the



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

Fig. 5: 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.

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. 5(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.

6.4 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. If the source and target models permits easy domain adaptations, the cloud no longer needs to tightly match the hardware configuration with the user device.

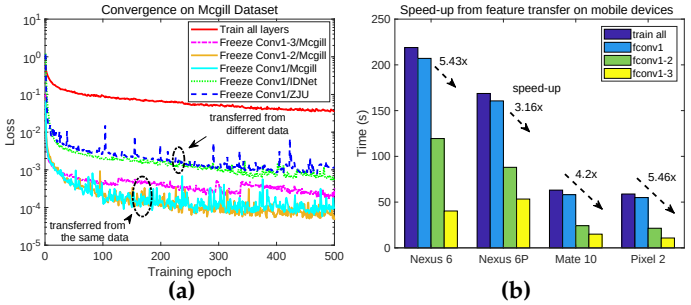
Fig. 6(a) shows the convergence of a random individual from the Mcgill dataset. We can see that feature transfer offers at least two orders of magnitude speed-up in terms of convergence. Features learned from data gathered with different settings offer significant boost as well. For instance, for the loss value to converge to 0.05, the original training 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. 6(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 3). 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.

6.5 Robustness against Intra-class Variations

We show that incorporation of training on mobile devices offers fast response to intra-class variation when behavioral

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 3: Accuracy with feature transfer



(a) (b)

Fig. 6: Boost from feature transfer (a) speed of convergence; (b) speed-up on mobile devices.

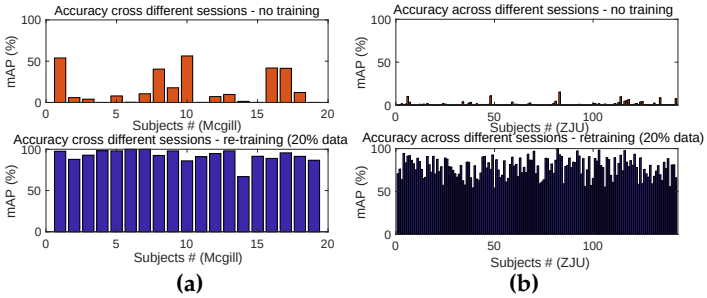


Fig. 7: Acceptance rate across different sessions (a) Mcgill; (b) ZJU.

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 4: Success ratio of passive attacks using Osaka dataset

biometrics evolve. We utilize the 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. 7, 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. 7 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.

6.6 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 4, 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.

6.7 Robustness against Active Attacks

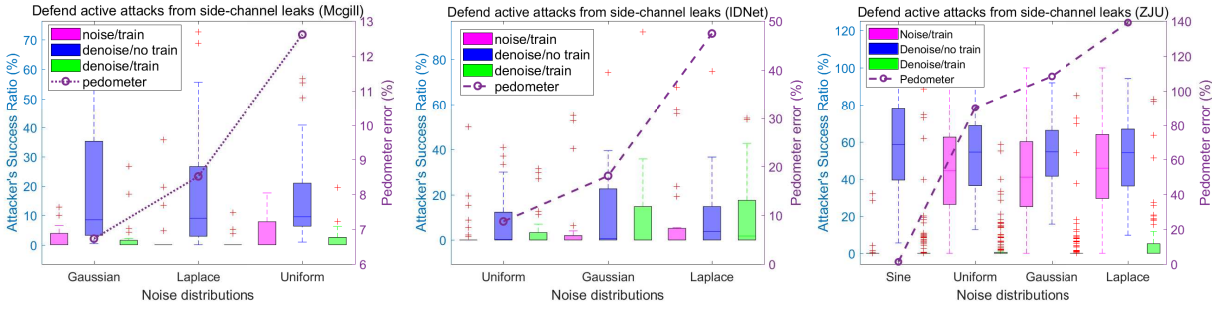
Next, we evaluate the system robustness against *active* attacks. Sec. 4.4 has shown that simple noise injection does not work well for obfuscation. In addition to *Gaussian* noise, we further evaluate *Laplacian* and *Uniform* noise with the standard deviation set to the original signal over a finite moving window. Laplacian noise is also used in differential privacy for mathematical tractability. We adopt the three types of noise to evaluate their properties regarding obfuscation and impact on usability. We choose a typical application of pedometer step counter to assess usability in the presence of noise. Fig. 8 shows the attacker's success ratio versus the pedometer error for different noise distributions. We alter the input in three ways. 1) *noise/train*: noise samples are paired with genuine ones in the training set and labeled as negative. 2) *denoise/no train*: attacker applies a state-of-the-art denoise technique called total variation proximity operators [40] on (1). The classifier takes no countermeasure. 3) *denoise/train*: the classifier makes a successful prediction about the denoise scheme and labels the denoise pairs as negative for training.

Fig. 8(a-b) indicate that the proposed mechanism is capable of defending against active attacks when the Siamese Network is supervised to learn the difference from the attack samples (noised or denoised). Learning the noised signals can drop the success rate from 50% to less than 10%. However, without considering possible denoise from the attacker in the classifier, there is still around 20% success rate even when the neural network has learned the noised signals. Once denoise is considered in training, the attacker can no longer succeed. For usability, the noise distributions incur 7-13% error for the step counter. IDNet gathered from a more vibrant environment has higher intra-class deviations. When the standard deviation of noise is set to as large as the original signal, the step counter is subject to a higher error rate.

An anomaly is ZJU in Fig. 8(c), in which body sensors of low sampling rate are used. The additive noise has much higher frequency thus is bound to be filtered out by the neural networks. The error of step counter is almost doubled due to the local peaks of the noisy spikes being mistakenly recognized as gait cycles. Using random noise, we do not see much security improvement but a sharp usability degradation. Instead of noise distributions with high frequency, we further test a sinusoidal wave with a low frequency (identical to the gait signal with a much smaller amplitude). The sine wave is merged into the sensing signal and difficult to extract since the oscillating frequency is kept as a secret. On the other hand, the new frequency components are evident enough to be recognized by the neural network through training. As shown in Fig. 8(c), the attacker's success ratio quickly drops to nearly zero for most of the 136 individuals in ZJU. Our new finding suggests that random noise is not always a good solution to balance security and usability. The actual obfuscation should be considered with respect to the types of data source. For a better balance of security and usability, obfuscation with hidden regularity can be considered as a signature.

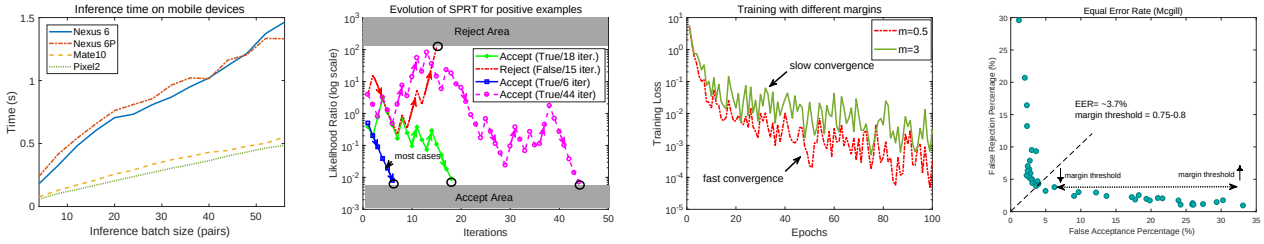
6.8 Other Important Metrics

This subsection evaluates factors that are equally important during implementation. Fig. 9(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



(a) (b) (c)

Fig. 8: Defend active attacks through side-channel leaks (a) Mcgill; (b) IDNet; (c) ZJU.

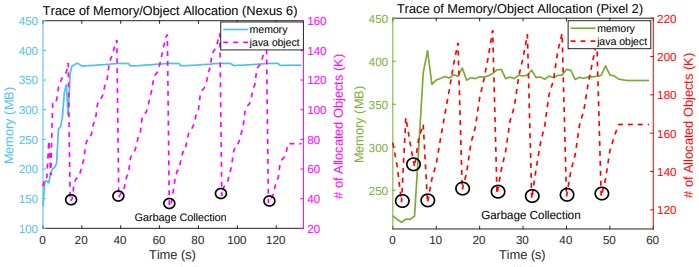


(a) (b) (c) (d)

Fig. 9: System metrics (a) batched inference time on mobile (b) SPRT for positive samples (c) speed of training convergence with different margins (d) equal error rate (Mcgill).

the input batch size. The computation takes less than 1.5s for all the devices. Table 4 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 SPRT as described in Section 4.3. Fig. 9(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 how the evolution fluctuates, 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, SPRT reduces authentication instability at a little cost of extended response time.

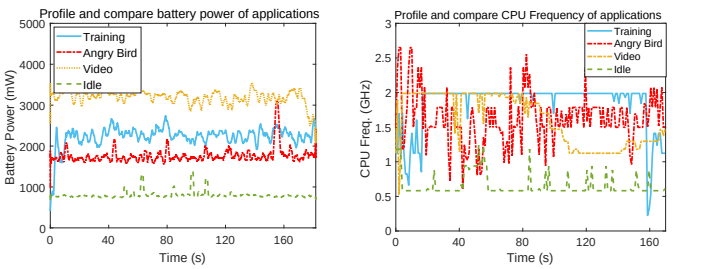
The authentication algorithm introduces a *margin* parameter in Eq. (1). It defines the boundaries between samples in high-dimensional feature space. In our testing, $m = 0.5$ maps the negative pair distance to around 2 and $m = 3$ maps it to 4.5. Intuitively, a small margin may lead to higher error rate because dissimilar pairs are closer in feature space and possibly misclassified as similar, or vice versa. A large margin makes it difficult to train the classifier in terms of slower convergence as shown in Fig. 9(c). For balancing the rates between false acceptance and false rejection, we set $m = 1.5$ and enumerate the margin threshold from 0.1 to 3 in Fig. 9(d). If the distance is below the margin threshold, the test pair is similar and dissimilar otherwise. For Mcgill, our framework achieves an equal error rate (EER) around 96.3% when the margin threshold is set to $\frac{1}{2}m$.



(a) **(b)**

Fig. 10: Trace of memory/object allocation during mobile training (a) Nexus 6; (b) Pixel 2.

6.9 Profile System Overhead



(a) (b)

Fig. 11: Profiling battery power and CPU frequency of different applications

Memory. We use the Android Profiler to measure the memory consumption of the app during training in Fig. 10.

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 network 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 [60] and CPU frequency by the Trepn Profiler [61]. 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. 11. 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.

7 Conclusion

This paper incorporates training on mobile devices and tackles the privacy and performance challenges when deep learning algorithms are migrated to resource-constrained battery-power devices. A comprehensive framework is designed to mitigate overfitting, side-channel leaks and maintain high authentication accuracy. The framework 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.

References

[1] M. Xu, J. Liu, Y. Liu, F. Lin, Y. Liu and X. Liu, "A first look at deep learning apps on smartphones", *WWW*, 2019.

[2] Low precision GEMM library, https://github.com/google/gemmlowp

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", *IEEE CVPR*, 2018.

[4] S. Han, H. Mao, W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding", *ICLR*, 2016.

[5] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science", *Nature Communications*, vol. 9, no. 2383, 2018.

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

[7] J. Ba and R. Caruana, "Do Deep Nets Really Need to be Deep?", *NIPS*, 2014.

[8] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang, "User verification leveraging gait recognition for smartphone enabled mobile healthcare systems," *IEEE TMC*, vol. 14, no. 9, pp. 1961-1974, 2015.

[9] W. Xu, G. Lan, Q. Lin, S. Khalifa, N. Bergmann, M. Hassan and W. Hu, "KEH-gait: towards a mobile healthcare user authentication system by kinetic energy harvesting",*NDSS*, 2017.

[10] P. Negi, P. Sharma, V. S. Jain, B. Bahmani, "K-means++ vs. behavioral biometrics: one loop to rule them all", *NDSS*, 2018.

[11] D. Liu, B. Dong, X. Gao, and H. Wang, "Exploiting eye tracking for smartphone authentication," *ACNS*, 2015.

[12] X. Li, Y. Zhang, I. Marsic, A. Sarcevic and R. Burd, "Deep learning for rfid-based activity recognition", *ACM Sensys*, 2016.

[13] M. Alsheikh, A. Selim and D. Niyato, L. Doyle and S. Lin and H. Tan, "Deep Activity Recognition Models with Triaxial Accelerometers", *AAAI Workshop*, 2016.

[14] S. Tople, K. Grover, S. Shinde, R. Bhagwan and R. Ramjee, "Privado: Practical and Secure DNN Inference", *arXiv preprint arXiv:1810.00602*, 2018.

[15] Q. Zhang, C. Wang, H. Wu, C. Xin and T. V. Phuong,"GELU-Net: a globally encrypted, locally unencrypted deep neural network for privacy-preserved learning", *IJCAI*, 2018.

[16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," *ICML*, 2016.

[17] X. Jiang, M. Kim, K. Lauter and Y. Song,"Secure outsourced matrix computation and application to neural networks", *ACM CCS*, 2018.

[18] M. Malekzadeh, R. G. Clegg and H. Haddadi, "Replacement autoencoder: a privacy-preserving algorithm for sensory data analysis," *IoTDI*, 2018.

[19] Z. Lu, S. Rallapalli, K. Chan and T. La Porta, "Modeling the resource requirements of convolutional neural networks on mobile devices", *ACM MM*, 2017.

[20] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints", *ACM Mobisys*, 2016.

[21] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars and L. Tang, "Neurosurgeon: collaborative intelligence between the cloud and mobile edge", *ACM ASPLOS*, 2017.

[22] E. Owusu, J. Han, S. Das, A. Perrig and J. Zhang, "ACCessory: password inference using accelerometers on smartphones", *ACM HotMobile*, 2012.

[23] Y. Michalevsky, D. Boneh and G. Nakibly, "Gyrophone: recognizing speech from gyroscope signals", *USENIX Security*, 2014.

[24] A. Das, N. Borisov, M. Caesar, "Tracking mobile web users through motion sensors: attack and defenses", *NDSS*, 2016.

[25] R. Ning, C. Wang, C. Xin, J. Li and H. Wu, "DeepMag : sniffing mobile apps in magnetic field through deep convolutional neural networks", *IEEE Percom*, 2018.

[26] S. Chopra, R. Hadsell and Y. LeCun, "Learning a similarity metric discriminatively with application to face verification", *IEEE CVPR*, 2005.

[27] G. Koch, R. Zemel, R. Salakhutdinov, "Siamese neural networks for one-shot image recognition", *ICML Deep Learning Workshop*, 2015.

[28] R. Shokri, V. Shmatikov, "Privacy-preserved deep learning," *ACM CCS*, 2015.

[29] X. Zeng X, K. Cao, M. Zhang,"MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images", *ACM Mobisys*, 2017.

[30] A. Mathur, N. Lane, D. Bhattacharya, S. Boran, A. Forlivesi, C. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware", *ACM Mobisys*, 2017.

[31] H. Khan, A. Atwater and U. Hengartner, "Itus: an implicit authentication framework for Android", *ACM Mobicom*, 2014.

[32] M. Gadaleta and M. Rossi, "IDNet: Smartphone-based gait recognition with convolutional neural networks", *Elsevier Pattern Recognition*, vol. 74, pp. 25-37, 2018.

[33] G. Hinton et. al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[34] H. Liu, S. Saroiu, A. Wolman, H. Raj, "Software abstractions for trusted sensors", *ACM Mobisys*, 2012.

[35] A. Serwadda and V. V. Phoha, "When kids' toys breach mobile phone security", *ACM CCS*, 2013.

[36] P. Shrestha, M. Mohamed and N. Saxena, "Slogger: smashing motion-based touchstroke logging with transparent system noise", *ACM WiSec*, 2016.

[37] J. Vitter, "Random sampling with a reservoir", *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37-57, 1985.

[38] A. Wald, "Sequential tests of statistical hypotheses," *Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 117-186.

[39] R. Shwartz-Ziv, N. Tishby, "Opening the black box of deep neural networks via information", *arXiv preprint arXiv*, 1703.00810, 2017.

[40] Total Variation proximity operator, https://github.com/albarji/proxTV

[41] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, "Understanding deep learning requires re-thinking generalization", *ICLR*, 2018.

[42] A. Shrivastava, A. Gupta and R. Girshick, "Training region-based object detectors with online hard example mining", *IEEE CVPR*, 2016.

[43] J. Yosinski and J. Clune and Y. Bengio and H. Lipson, "How transferable are features in deep neural networks?", *NIPS*, 2014.

[44] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them", *IEEE CVPR*, 2015.

[45] B. Hitaj, G. Ateniese and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning", *ACM CCS*, 2017.

[46] S. Mazilu, A. Calatroni, E. Gazit, A. Mirelman, J. M. Hausdorff and G. Trster, "Prediction of freezing of gait in Parkinson's from physiological wearables: an exploratory study", *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 6, pp. 1843-1854, Nov. 2015.

[47] S. Radovanović, M. Jovičić and N. Marić and V. Kostić, "Gait characteristics in patients with major depression performing cognitive and motor tasks while walking", *Psychiatry research*, vol. 217, no. 1-2, pp. 39-46, 2014.

[48] L. Maaten and G. Hinton, "Visualizing data using t-SNE", *Journal of machine learning research*, pp. 2579-2605, 2008.

[49] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *in Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556, 2014.

[51] Tensorflow Lite, https://www.tensorflow.org/lite/

[52] Caffe2 for iOS/Android, https://caffe2.ai/docs/mobile-integration.html

[53] MXNet for smart devices, https://mxnet.incubator.apache.org/faq/smart_device.html

[54] Deep Learning for Java, https://deeplearning4j.org

[55] McGill Dataset, http://jwf.github.io/Humansense-Android-App

[56] IDNet Dataset, http://signet.dei.unipd.it/human-sensing

[57] Y. Zhang, G. Pan, K. Jia, M. Lu, Y. Wang, Z. Wu, "Accelerometer-based gait recognition by sparse representation of signature points with clusters," *IEEE Transactions on Cybernetics*, vol. 45, no. 9, pp. 1864-1875, Sept. 2015.

[58] T. Ngo, Y. Makihara, H. Nagahara, Y. Mukaigawa and Y. Yagi, "The largest inertial sensor-based gait database and performance evaluation of gait-based personal authentication," *Pattern Recognition*, vol.47, no. 1, pp. 222-231, 2014.

[59] Y. Wen, K. Zhang, Z. Li and Y. Qiao, "A discriminative feature learning approach for deep face recognition", *European Conference on Computer Vision*, Springer, 2016.

[60] Monsoon Power Monitor, https://www.msoon.com/online-store

[61] Trepn Power Profiler, https://developer.qualcomm.com/software/trepn-power-profiler