DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks

Rui Ning, Cong Wang, ChunSheng Xin, Jiang Li, and Hongyi Wu
Old Dominion University, Norfolk, VA 23529, USA

Abstract— **In this paper, we report a newfound vulnerability on smartphones due to the malicious use of unsupervised sensor data. We demonstrate that an attacker can train deep Convolutional Neural Networks (CNN) by using magnetometer or orientation data to effectively infer the Apps and their usage information on a smartphone with an accuracy of over 80%. Furthermore, we show that such attacks can become even worse if sophisticated attackers exploit motion sensors to cluster the magnetometer or orientation data, improving the accuracy to as high as 98%. To mitigate such attacks, we propose a noise injection scheme that can effectively reduce the App sniffing accuracy to only 15% and at the same time has negligible effect on benign Apps.**

I. Introduction

Smartphones have become constant companions in our daily life. People are not just using their mobile devices at work and home – they are living on them. People rely on smartphone applications (Apps) for communication, social networking, entertainment, banking, shopping, navigation, healthcare, and more. For many people, every day begins and ends with checking their smartphones. As more and more personal data are stored on and processed and transmitted by smartphones, they are becoming an increasingly attractive target for cybercriminals. For example, recent studies have shown several attacks by exploiting smartphone sensors [1]–[5].

The modern smartphones integrate a diversity of sensors for productivity, such as accelerometer, gyroscope, magnetometer, GPS, gravity sensor, barometer, microphone, ambient light sensor, and proximity sensor. The smartphone operating systems often create their own access control for the sensors. Some sensors, for instance, the location sensor (GPS) and audio sensor (microphone) trigger privacy concerns and hence require user permission before any App can access their data. On the other hand, a range of sensors (including accelerometer, gyroscope, and magnetometer), are called unsupervised sensors, which are not considered explicitly sensitive and thus accessible freely by Apps. Some of them can be even obtained by Javascript embedded in webpages [3].

In this paper we report a newfound vulnerability on smartphones due to the malicious use of unsupervised sensor data, where the attacker can sniff mobile Apps, i.e., infer what Apps have been installed on the user's mobile device, how frequently they are used, and when they are opened, by analyzing the magnetometer data along with the motion sensor data using deep learning techniques. The accuracy can be as high as 98%.

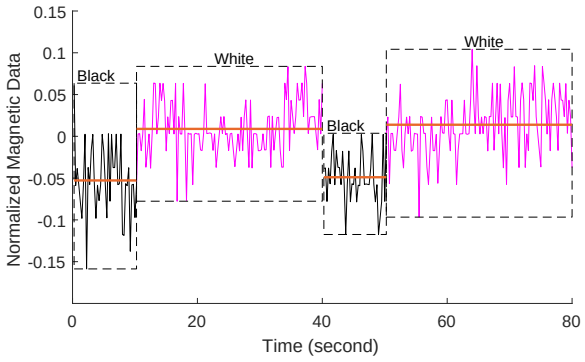
A. Preliminary Experiments and Observations

The quest begins with the observation of a subtle correlation between an LED display and its surrounding magnetic field. For example, in our first experiment, we display a black image on a 27 inch LED PC monitor for 20 seconds, followed by a white image for 60 seconds. We repeat the pattern for a number of rounds. In the meantime, an iPhone 7 Plus is placed in front of the monitor about 10*cm* away to measure the magnetic field. We observe a noticeable change in the magnetic field while switching between the two images (see Fig. 1(a)).

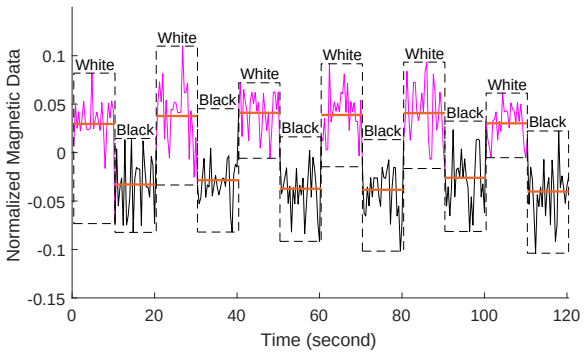
The above phenomenon motivates us to further explore how the LED display on a smartphone would affect its magnetometer readings. To this end, we carry out a similar experiment by displaying the black and white images on the iPhone 7 Plus while recording the data captured by the magnetometer on the same phone. Compared with the previous setting, we expect more stable results since the distance between the LED display and the magnetometer is shorter and their relative orientation is fixed. The experimental data are depicted in Fig. 1(b), demonstrating significant changes in magnetic field when different images are displayed on the phone.

Fig. 1(c) further shows the change of magnetometer readings while the smartphone displays four different colors, white-black-red-blue, in sequence. While it is beyond the scope of the paper to fully model this physical phenomenon, it is largely due to the fact that different bias voltages are used when LED displays different colors, which accordingly lead to the change of the magnetic field [6].

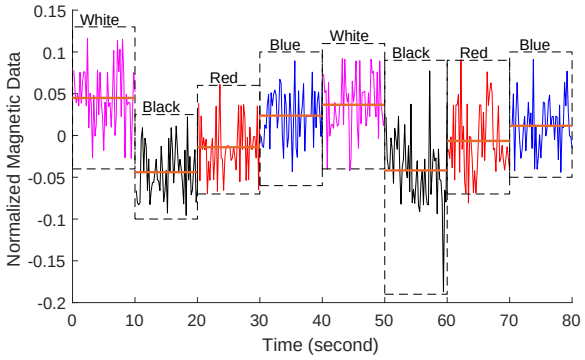
The results shown in Fig. 1 demonstrate the correlation between colors on LED display and surrounding magnetic field. Since different Apps often adopt different graphic designs that mix different color patterns, we speculate that they also induce different magnetometer readings. In particular, when one clicks on an App's icon, a unique welcome-page will be displayed till the App is fully open. The corresponding changes in magnetic field can be measured by the integrated magnetometer. Fig. 2 illustrates the averaged magnetometer readings over the period for opening two popular Apps, i.e., Snapchat and Twitter, on an iPhone 7 Plus. As can be seen, they differ dramatically on at least one axis (in this case, Y-axis). This is because Snapchat adopts predominantly a bright yellow color while Twitter uses blue. We have



(a)



(b)



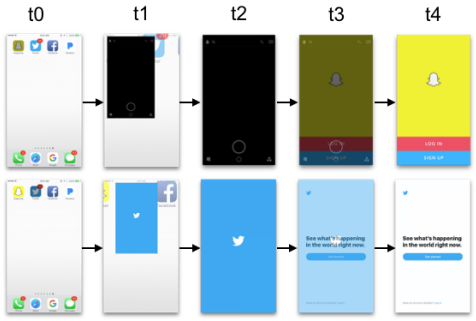
(c)

Fig. 1. Magnetometer readings on Y-axis. (a) Change of magnetometer reading due to PC LED display, where the black and magenta waveforms correspond to the black and white images, respectively. (b) Change of magnetic field due to smartphone LED display (black and white). (c) Change of magnetic field while the smartphone sequentially displays white-black-red-blue. The solid orange line in each dashed rectangle box represents the mean of the signal within the box.

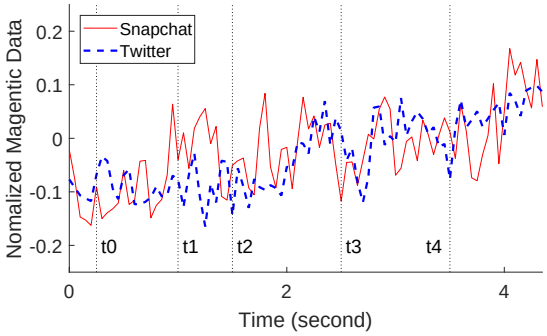
verified that the above observations are repeatable on different smartphones and models. More results will be presented in Secs. II-IV. These preliminary experimental data indicate that different Apps are likely associated with unique magnetic signatures. Therefore, if one can acquire magnetometer data, he can potentially infer the Apps running on a smartphone.

B. Our Contributions

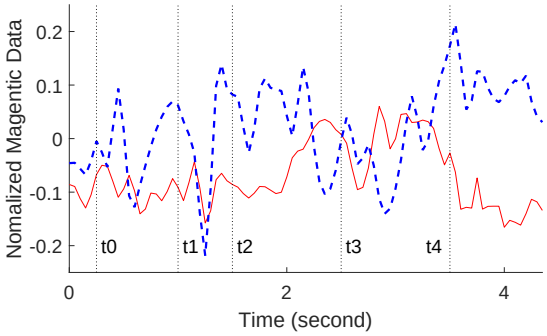
This is the first work that discovers and reports the correlation between magnetometer readings and LED displays on smartphones. Based on this observation, we demonstrate a newfound side-channel attack, where the attacker can sniff mobile Apps through magnetometer data. In particular, we devise deep Convolutional Neural Networks (CNN) that can be trained to effectively infer



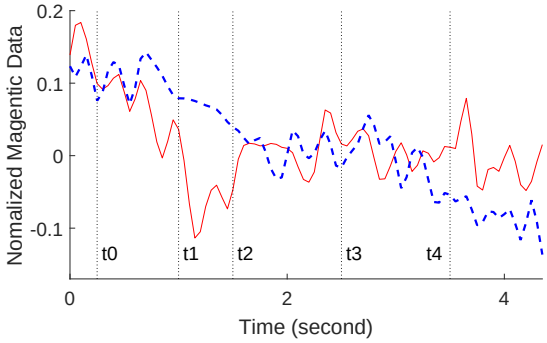
(a) Screen shots when opening Snapchat and Twitter.



(b) Magnetometer X-axis readings for Snapchat and Twitter.



(c) Magnetometer Y-axis readings for Snapchat and Twitter.



(d) Magnetometer Z-axis readings for Snapchat and Twitter.

Fig. 2. Welcome pages and magnetometer readings of popular Apps.

the Apps installed on the smartphone and their usage information. We implement the attack on iPhone 7 Plus and Samsung Galaxy 7 and carry out extensive experiments, showing that the attack can achieve an accuracy of about 84% based on the magnetometer data.

Furthermore, we discover that the orientation data is closely correlated with the magnetometer readings. To this end, we use the orientation of the smartphone as an alternative to train and test the CNN models. The performance is only slightly lower than the original approach based on magnetometer data. This finding enables even more pervasive attacks since the orientation data can be readily obtained by integrating a 4-line Javascript code in attacker's websites and the Javascript can continuously acquire the orientation data even in the background.

Moreover, we show that the attack can become worse if the attacker exploits motion sensor data. Briefly, for a given mobile phone, the locations of the Apps are generally fixed on the screen. Suppose the mobile user clicked on an App for k times during a period. If the k magnetometer or orientation data are fed into the CNN model, about 16% of them would be misclassified to other Apps. However, if the attacker is able to recognize that the k clicks are all at the same location on the screen, then he can put them into a cluster and feed the cluster of magnetometer or orientation data to the CNN model. The vast majority of them (about 84%) should be recognized correctly. Since the cluster of clicks are from the same location, they should be the same App. Therefore, a majority vote can effectively determine the App for the entire cluster. This Approach can increase the accuracy to as high as 98%.

Besides showing this newfound side-channel attack, we also discuss viable methods to mitigate it by injecting minimal amount of noise into the magnetometer or orientation data. The rest of this paper is organized as follows. Sec. II introduces the magnetometer-based App sniffing, including extended discussions on attacks based on orientation data. Sec. III demonstrates the motion sensor-assisted approaches. Sec. IV presents experiments and results. Sec. V discusses viable schemes to defeat the attacks. Finally, Sec. VI concludes the paper.

II. Magnetometer-based App Sniffing (MAS)

In this section, we present App sniffing solely based on the readings from magnetometer. The preliminary observations presented in Sec. I indicate that different Apps are likely to induce different magnetic field. However, it remains a nontrivial problem to identify the unique magnetic signature for each App and accordingly infer the Apps according to magnetometer readings. The fundamental challenge lies in the facts that the magnetometer data exhibit noise and randomness and that the Apps' graphic designs often incorporate complex combination of color patterns, rendering simple classification methods infeasible. To this end, we propose to exploit the powerful deep *CNN* to classify magnetometer data and to infer the corresponding Apps.

A. Deep CNN Models for Magnetometer-based App Sniffing

Magnetometer data can be recorded when a user opens an App. In our preliminary exploration, we have considered the top-7 most used Apps, i.e., Twitter, Snapchat, Pandora, Netflix, Google Maps, Chase Bank, and HBO. The recording process essentially collects a sequence of magnetometer samples during the interval from clicking on an App to the time when the welcome page of the App is fully displayed on the mobile screen. Fig. 2 shows the examples for opening Snapchat and Twitter, respectively. Different phones may have different sampling rates of their magnetometers. For instance, Samsung Galaxy S7 and iPhone 7 Plus sample their magnetometers at the rate of 20Hz and 100Hz, respectively. As to be shown later, the sampling rate has negligible impact on the effectiveness of App sniffing.

Since the change of magnetic field is relatively small, we preprocess the raw magnetometer data by using a denoising function (e.g., *wden* available in MATLAB [7]). It decomposes the signal into wavelets and performs thresholding on wavelets coefficients. We set the function at the denoise-level 5 with soft thresholding rule for the universal threshold $\sqrt{2\ln(\cdot)}$. Then, we normalize the denoised data by subtracting its mean and dividing it by its vector's norm. Figs. 2(b)-(d) illustrate the normalized magnetometer readings of Snapchat and Twitter on X, Y, and Z axis. While the figure only shows about 4 seconds, the total recording time is 8.25 seconds including some overhead before and after the App is opened. Thus on an iPhone with a sampling rate of 100Hz, each magnetometer data would have a dimension of 825×3. Directly feeding such high dimensional data into the CNN yields poor results (less than 50% accuracy) because drastic changes over such a large span of 825 sample points may easily overwhelm a neural net's representational capability. To this end, we adopt a sliding window approach with a window size of W sample points to slide over the time series. Each pair of adjacent slices has an overlap of P sample points. For example, assume $W = 36$ and $P = 31$. If we have 100 original data (each with a dimension of 825×3), they will be converted to $15,800$ sliced data each with a dimension of 36×3. The sliced data are labeled with corresponding Apps for training and testing as to be discussed next.

The success of CNN has been demonstrated in computer vision [8]–[10]. Compared to traditional learning techniques based on hand-crafted features, CNN can be trained from end-to-end to extract features automatically. It usually stacks multiple convolutional layers, each comprising filters to capture spatial patterns in the data and activations that represent the result of convolution. CNN exploits these layered structures of non-linearities to characterize highly complicated relations in the data.

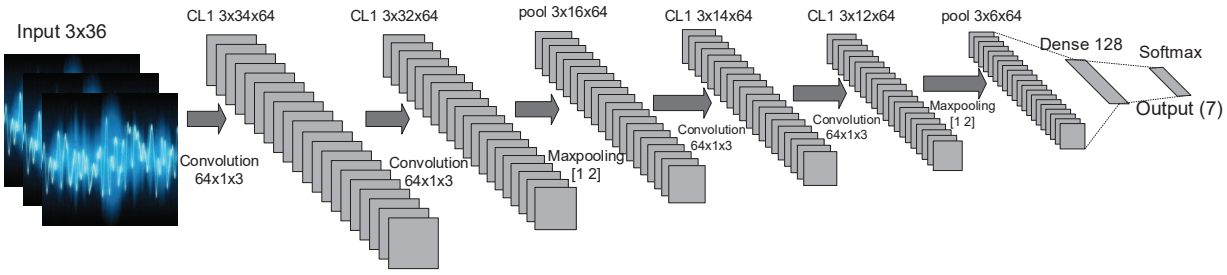


Fig. 3. An example of the proposed deep CNN architectures.

Max pooling layers are usually introduced between convolutional layers to reduce the number of parameters and speed up computation.

We have explored several deep CNN architectures to sniff Apps based on magnetometer data. In contrast to computer vision (that adopts 2D filters for images), the magnetometer data on smart phone involves 3 channels (x, y, z) and data points along each channel dimension is a 1D time series. To this end, 1D filter is adopted in the architectures to capture temporal correlations on each channel. Similar approaches are also found in human activity recognition [11], [12].

Our goal is to demonstrate that an appropriately designed CNN can effectively sniff frequently used Apps on a mobile phone. The exploration begins with a 3-Layer architecture consisting of only one convolutional layer. We adopt 1×3 kernels that capture subtle changes along the time axis. Each layer applies *Rectified Linear Units* (ReLUs) as the activation functions (taking $f(x) = \max(0, x)$). The features extracted by convolutional layers are fed into a densely connected layer which connects to the output softmax function.

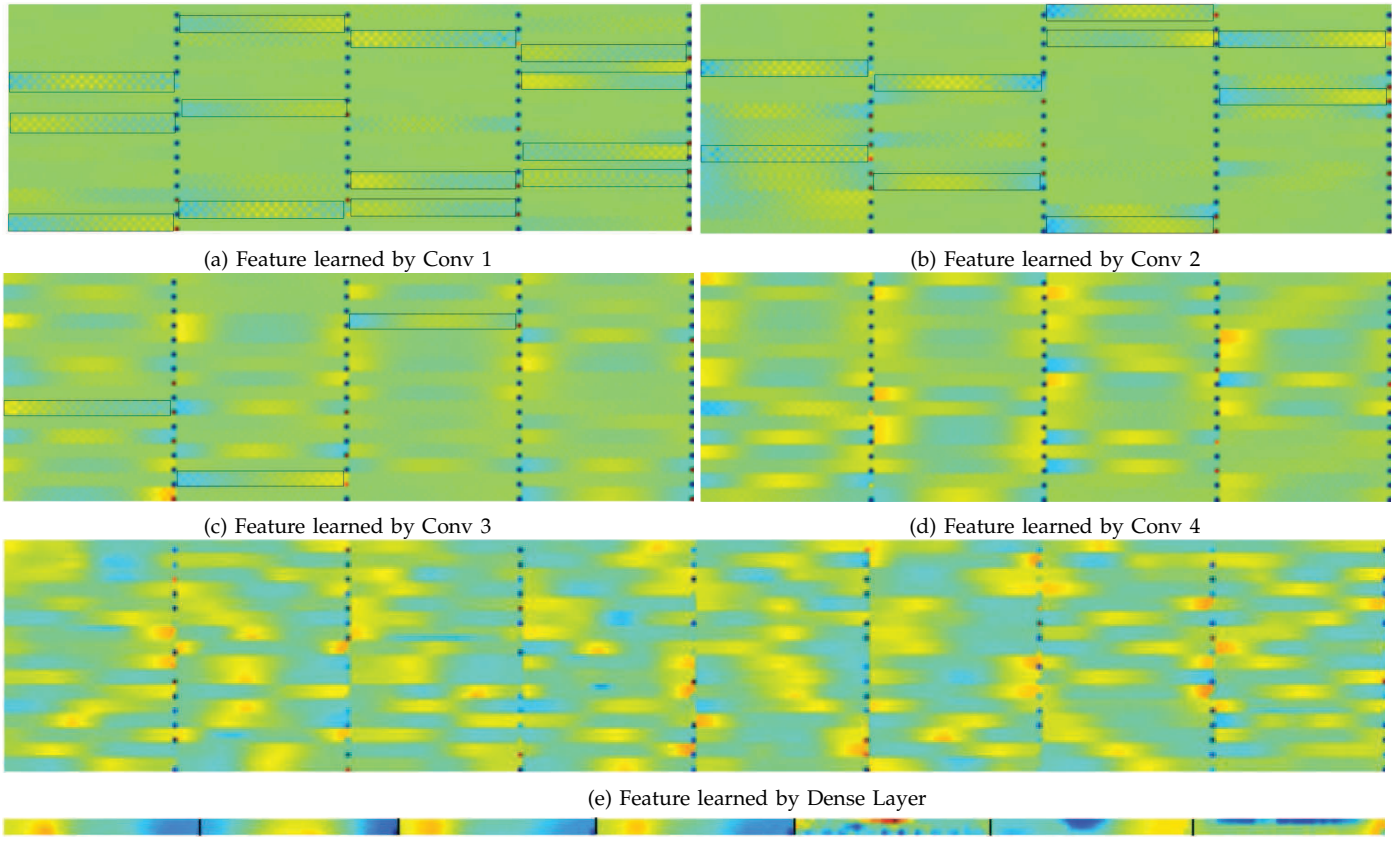
Although a single convolutional layer is fast for computation, low-level features captured in the first layer may not generalize well on the entire dataset. To exploit the wealth of data that an attacker can obtain, we have further investigated several deeper structures by stacking more convolutional layers together and inserting max pooling layers to reduce dimensionality. Fig. 3 illustrates an example of such deep CNN structures. For brevity, a 6-layer CNN is denoted as: *Conv*(64)-*Conv*(64)-*Pool*-*Conv*(64)-*Conv*(64)-*Pool*-*Dense*(128)-*Sfmax*. A layer is counted if it has adjustable weighted connections. Each convolutional layer has 64 filters and the densely connected layer has 128 neurons with ReLU activations. Maxpooling layer reduces the input dimension by half. As more convolutional layers are stacked up, the network will be able to extract high-level features and generalize on the dataset.

To understand the effectiveness of this CNN approach and compare the accuracy of different CNN architectures, we have carried out a set of preliminary experiments. We have considered the top-7 most used Apps as discussed earlier (from Twitter to HBO), and collected a total of 700 raw magnetometer recordings on a number of Samsung Galaxy S7 and iPhone 7 Plus units. They are the flagship smartphones of Samsung and Apple – two companies that together have a total market share of 72.8% in the US [13]. The CNN models have been implemented in *Tensorflow* [14] with batch sizes of 150 and 100 epochs. For comparison, we have also implemented a baseline 3-layer neural network (NN) model that has dense connections and a support vector machine (SVM) model using LibSVM [15]. All of them are trained and tested on a PC with I7-4470 CPU and NVIDIA 1080Ti graphic card. As discussed earlier, we adopt a sliding window approach to slice each recorded magnetometer data. The default parameters are $W = 36$ and $P = 31$.

The primary performance metric is the accuracy, i.e., the fraction of correctly recognized Apps. We utilize 4-fold cross validation (CV) for performance evaluation, where we randomly divide a dataset to 4 parts and use three parts for training and the remaining part for testing. This process is repeated four times such that each part is used for testing once. We are also interested in the running time. For a 6-layer CNN model, the training time for each epoch is around 4 seconds, and it takes about 100 epochs to achieve converged result.

As shown in Table I, the 6-Layer CNN has the highest accuracy. At the same time, we also observe that the accuracy is not sensitive to different CNN architectures. All of them perform significantly better than SVM and NN. Thus, an attacker can utilize any general-purpose CNN to construct the attack without the need to fine-tune the CNN model.

Fig. 4 visualizes the features learned on input signal captured by the 4 convolutional layers, dense layer and finally the output layer. Each convolutional layer trains a number of filters to match similar spatial patterns in the input signal in order to minimize the cost function. Each learned feature is displayed as 3×36 (size of the signal input) and arranged in a stitched $16 \times 4 = 64$ array for each layer. Here, 3 is the number of channels and 36 is the sliding window size. Since the features learned from the signals are rather flat, we remove the margins between neighboring features for better visualization. From Fig. 4(a), the highlighted features (boxed) show mosaic patterns that are activated from the raw input



(a) Feature learned by Conv 1

(b) Feature learned by Conv 2

(c) Feature learned by Conv 3

(d) Feature learned by Conv 4

(e) Feature learned by Dense Layer

(f) Feature learned at Output Layer (Prediction)

Fig. 4. Visualization of features learned by the deep CNN.

signal. Subsequent layers are more abstract and such low-level, mosaic patterns start to disappear from the 3rd layer. Although the high level features learned by the CNN are not visually explainable at this point [16], they jointly represent unique identifications for each class of Apps on the high level. The learned features from convolutional layers are fed into the dense layer that classifies the outputs into 7 classes. The output layer is a generalization of all the features learned by the network that average over the data points in each class. This is consistent with the reasoning in [17]. In contrast, SVM and NN fail to effectively identify different Apps. It may be due to the fact that the time series from different Apps are overlapped and these models lack the automatic feature learning capabilities.

We further compare the performances by tuning different hyperparameters. The experimental results (omitted here due to space limit) show that the model is not sensitive to the batch size. The slice window with $W = 36$ and $P = 31$ always yields the highest accuracy. We also observe that the errors are generally evenly distributed unless two Apps are very similar in colors and patterns. As shown in Table II, the confusion between HBO and Netflix is relatively high, i.e., most errors of Netflix are misclassified into HBO, and vice versa, because their colors are both predominantly black. This is also reflected in Fig. 4(f), where the 1^{st} and 4^{th} classes, which respectively correspond to HBO and Netflix, show similarities.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT CNN MODELS.

Machine Learning Model	Accuracy
SVM	0.3923
SVM with Sliced Input Data	0.4214
3-Layer NN	0.4333
3-Layer NN with Sliced Input Data	0.4577
3-Layer CNN with Sliced Input Data	0.8198
5-Layer CNN with Sliced Input Data	0.8273
6-Layer CNN with Sliced Input Data	**0.8389**
7-Layer CNN with Sliced Input Data	0.8317
8-Layer CNN with Sliced Input Data	0.8368

TABLE II
CONFUSION MATRIX (UNDER 6-LAYER CNN WITH SLICED INPUT).

	HBO	Chase	Google Maps	Netflix	Pandora	Snapchat	Twitter
HBO	70.13	1.71	0.27	14.92	2.85	1.34	1.41
Chase	3.78	85.54	3.26	1.09	1.12	2.26	2.32
Google Maps	3.33	2.1	86.53	1.46	1.23	1.34	1.43
Netflix	16.56	1.26	3.64	72.79	1.48	1	0.37
Pandora	0.7	3.52	0.68	2.45	89.12	1.19	3.02
Snapchat	3.14	2.24	1.17	2.94	2.86	91.81	0.12
Twitter	2.36	3.63	4.45	4.35	1.34	1.06	91.33

B. Orientation-based App Sniffing

The previous subsection has shown a possible approach to sniff Apps on a mobile phone based on

magnetometer data. The accuracy is about 0.84. We will introduce enhanced schemes which achieve a higher accuracy of close to 1 in the next section. But before that, we would like to discuss how an attacker can obtain sensory data from a user's smartphone.

By default, any App on a smartphone can access magnetometer without user permission. So the most straightforward approach is to embed a small piece of code in an benign App to report magnetometer data to the attacker. However, not all mobile users would install such App. Toward this end, we have further considered a web-based method, where the attackers can acquire sensor data by simply integrating a small (4 line) Javascript code in their webpage. When a smartphone browses the webpage, the Javascript can read a range of sensory data such as gyroscope and accelerometer that we will use later. However, it can not attain direct access to the magnetometer of the mobile devices.

```
function deviceOrientationHandler(eventData) {
  var ori_gamma = eventData.gamma;
  var ori_beta  = eventData.beta;
  var ori_alpha = eventData.alpha;
}
```

Nevertheless, our investigation reveals that the orientation of the devices can be sniffed using the web-based method. Furthermore, the orientation is closely correlated with the magnetometer data as shown by comparing Fig. 1(c) and Fig. 5. As a matter of fact, the primary use of magnetometer data on the smartphone is to calculate the device's orientation in conjunction with the accelerometer. Based on this interesting observation, we use the orientation of the device as an alternative to train and test the CNN model. The performance (i.e., recognition accuracy) is only slightly lower than the original approach based on magnetometer data. Details results will be discussed in Sec. IV (see Table IV and related discussions).

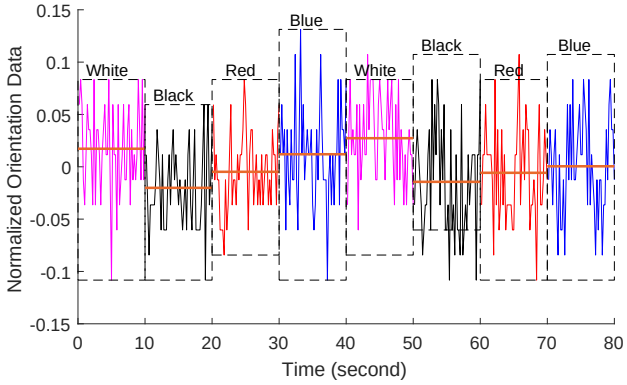


Fig. 5. Change of orientation data while the smartphone sequentially displays four different colors, white-black-red-blue.

III. Motion Sensor-Assisted MAS

In the previous section, we have demonstrated a side channel attack, i.e., MAS, where the attacker sniffs the magnetometer data on a smartphone and accordingly infers the Apps opened by the mobile user. The accuracy of such inference is about 84%. In this section, we show that the attack can be worse, i.e., the attacker can achieve even higher accuracy, if he exploits motion sensor data.

Briefly, on a given smartphone, the locations of the Apps are generally fixed during the time window when the attacker sniff sensor data (e.g., ranging from a few hours to a few days). Suppose the mobile user clicked on Chase App 20 times during the period. If the 20 magnetometer or orientation data are fed into the CNN model introduced in Sec. II, about four of them would be classified incorrectly to some other Apps. However, if the attacker is able to recognize that the 20 clicks are all at the same location on the screen, then he can put them into a cluster and feed the cluster of magnetometer data to the CNN model. The vast majority of them (about 84% in average) should be recognized correctly as Chase. Since the cluster of clicks are from the same location, they should be the same App.[1] Therefore, the attacker can conclude that all 20 clicks are Chase. This approach is effective because, as to be shown next, such clustering can be achieved with high accuracy (nearly 100%) by using motion sensor data on the phone.

A. Clustering based on Motion Sensor Data

Nowadays, most mobile operating systems place their App icons in a fixed grid layout. For instance, iPhone 7 plus has a 4x7 layout and Samsung S7 uses a 4x5 grid. When a user clicks on different spots on the touch screen, the smartphone has a small rotation and/or vibration that can be captured by the 3-axis gyroscope and accelerometer. Similar to the discussion on magnetometer, different phones sample their motion sensors at different frequencies: 100 Hz on iPhone and 50 Hz on Samsung Galaxy. Fig. 6 illustrates the 3-axis gyroscope data while clicking on the top-left and bottom-left on an iPhone's screen. As can be seen, the gyroscope waveforms differ on all three axes, especially the X and Z axes. Several previous studies have shown the use of motion sensors to infer the touches on a smartphone screen [18].

The attacker can build a training data set for each popular smartphone model and feed the training data to CNN to create a motion classifier for this type of smartphone. Note that, we cannot mix the training data of iPhone and Samsung phone since their layout are different. Again, various CNN architectures and hyperparameters are explored, and finally we adopt a 4-layer CNN model with the following setting in our experiments: *Conv*(64)-*Conv*(64)-*Pool*-*Dense*(128)-*Dropput*(0.5)-*Sfmax*. As discussed earlier, motion sensor (including accelerometer and gyroscope) data are freely accessible by Apps or web-embedded Javascript. For

[1]We have assumed that the most frequently used Apps are on the home page. The scenarios with multiple pages and groups will be investigated in our future research.

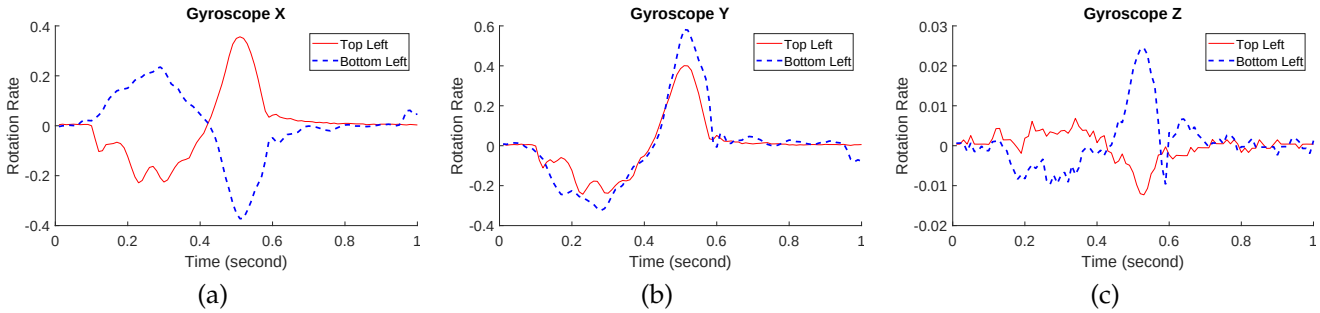


Fig. 6. Comparison of gyroscope data on X, Y, and Z axis while clicking on top-left and bottom-left of the screen.

training purpose, the attacker can easily experiment on different phones to build training data sets. For example, given a type of phone, the attacker can click on every grid points for a number of times, collect motion sensor data, and label each data with the corresponding grid point. The 3-axis gyroscope data and accelerometer data are combined together. A click lasts about 1 second. So, each data has a dimension of 6×100 on iPhone since its sampling rate is 100 Hz, or 6×50 on Samsung S7 that samples at 50 per second.

The CNN models are trained offline. Once they are ready, the attacker can acquire motion sensor data from mobile users either via Apps installed on the users' phones or when the mobile users browse the attacker's websites embedded with his Javascript as we discussed earlier. The data from each user are fed into the selected CNN model that matches the user's phone. If the type of phone is unknown, the attacker can always try different CNN models and choose the one that yields the most reasonable result. Thus, the attacker can label each click with a grid position on the screen, and accordingly group all clicks with the same label into a cluster.

For example, in our preliminary experiment, we have collected gyroscope and accelerometer data when clicking 100 times on each spot of an iPhone 7 Plus's screen. The experiments have been repeated by two people using both right and left hands. In total, dataset contains $2 \times 2 \times 100 \times 4 \times 7 = 11200$ data samples and each data sample has a dimension of 6×100. Again, 4-fold cross validation is adopted. Table III shows the clustering results. When we only consider four possible positions (at the four corners of the screen), the accuracy is perfectly 1.0. With the increase of grid points, the accuracy decreases slightly but is still maintained stable around 98%. With such high accuracy, the attacker can effectively group the clicks into clusters and consider all clicks in the same cluster to be associated with the same App.

B. Motion Sensor-Enhanced Attack

Based on the above findings, we now combine the clustering scheme (enabled by motion sensors) and the App classification based on magnetometer or orientation data. The overall procedure of the motion sensor-assisted MAS is illustrated in Fig. 7.

TABLE III
PERFORMANCES OF CLUSTERING BASED ON MOTION SENSORS.

No. of Grids	4	8	12	16	20	24	28
Accuracy	1.000	0.996	0.992	0.990	0.980	0.979	0.978

Assume that a smartphone has visited the attacker's website embedded with the aforementioned Javascript. Since the Javascript has free access to accelerometer, gyroscope and orientation, the attacker can continuously eavesdrop such sensor data. Note that, even the webpage is in the background, the Javascript can still acquire data. Of course, the attacker can also try to camouflage codes in seemingly benign Apps, given the behavior of accessing the sensor data is fairly common.

The hacker continuously collects sensor data for a desired period (usually for several hours or days), and then performs a straightforward preprocessing to identify the clicks on the screen and format corresponding motion data and magnetic (or orientation) data as follows:

$$\begin{pmatrix} click_1 & motion_1 & magnetic(orientation)_1 & time\_stamp_1 \\ click_2 & motion_2 & magnetic(orientation)_2 & time\_stamp_2 \\ \vdots & \vdots & \vdots & \vdots \\ click_n & motion_n & magnetic(orientation)_n & time\_stamp_n \end{pmatrix}$$

Each row represents one data comprising a click ID, corresponding motion sensor recording, magnetic or orientation recording, and other information such as the time stamp. The attacker first uses the 4-layer motion CNN model to classify each click (i.e., each row of the dataset) into a cluster and label it with the corresponding grid ID. The maximum number of clusters equals to the number of grids on the mobile screen. Note that, all clicks in the same cluster are from the same grid location and thus should be the same App. As shown in Table III, this step is very accurate. Next, the attacker feeds a cluster of magnetic (or orientation) data with the same grid ID to the 6-layer magnetic CNN model. Over 80% of them are expected to be classified correctly, according to the accuracy presented in Table I. Because they all belong to the same App, a majority vote can effectively determine the App for the entire cluster.

As a result, the attacker can infer what Apps have been installed on the user's mobile device, how frequently

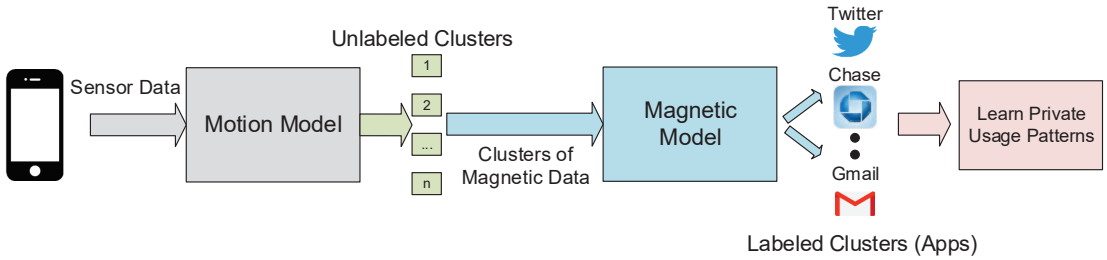


Fig. 7. The overall procedure of the motion sensor-assisted MAS.

they are used, and when they are opened. In other words, the attacker can track the users' habits of App usage. The attack can become even worse. For example, if the attacker has identified a bank App, it is not too hard to capture the motion sensors while the user types username and password. If the keyboard is static, the attacker can obtain the password by using a similar approach as discussed above and access the bank account.

IV. Experimental Evaluation

We have used iPhone 7 Plus and Samsung Galaxy 7 to implement and demonstrate the attacks. In this section, we present our experiments and results.

A. System Setup

To collect motion sensor training data, we have considered iPhone 7 Plus and Samsung S7 separately, since they have different App layouts: 4×7 and 4×5. For each grid point in a layout, we have collected 400 samples. The corresponding dataset size for each model is $400 \times 4 \times 7$ and $400 \times 4 \times 5$.

To recognize Apps, we have considered both magnetometer and orientation data. As discussed before, the latter is desired because it can be accessed by Javascript embedded in the attacker's website. To create the training dataset, we have considered top-15 most used Apps. We open each App 100 times and record the corresponding magnetometer and orientation data.

All CNN models are implemented in *Tensorflow* [14]. The training is done offline and the training data can be easily obtained by the attacker himself. On the other hand, it is trickier to collect sensor data of the victims. To this end, we have developed a mobile webpage embedded with a sensor data collection Javascript (for gyroscope, accelerometer, and orientation data) and a sensor data collection App on both iOS and Android (for gyroscope, accelerometer, and magnetometer data). We run the experiment for one day to collect data from users. Some clicks are to open Apps while others happen in the Apps. We differentiate them by detecting the pressing of the home button on iPhone or Samsung phones. We only process the sensor data of the clicks after pressing home button. Note that, the attacker is not necessary to process all clicks. As long as he can collect enough number of useful sensor data, he can achieve his goal of sniffing users' Apps and tracking their usage.

TABLE IV
Performance Comparison of MAS Approaches.

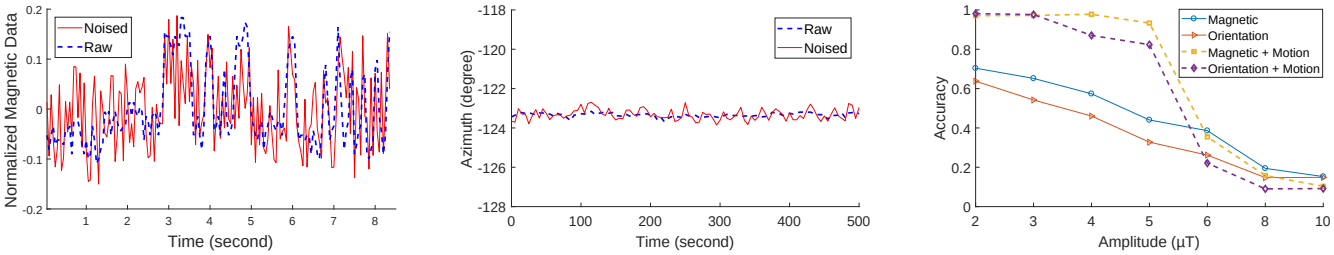
Number of Apps	3	7	11	15
Magnetic Model (Single Device)	0.8913	0.8389	0.8120	0.7916
Magnetic Model (Cross Device)	0.8979	0.8249	0.8157	0.7882
Magnetic Model (Cross Model)	0.6834	0.6233	0.5816	0.5314
Magnetic Model (Cross Model Mix)	0.8809	0.8244	0.7975	0.7653
Magnetic Model (Cross Model Mix) + Downsampling	0.8869	0.8313	0.8033	0.7793
Magnetic Model (Cross Model Mix) + Upsampling	0.8885	0.8311	0.8061	0.7846
Orientation Model (Cross Model Mix)	0.8279	0.7849	0.7557	0.7174
Magnetic Model (Cross Model Mix) + Motion	0.9786	0.9833	0.9767	0.9753
Orientation Model (Cross Model Mix) + Motion	0.9706	0.9755	0.9895	0.9806

We have shown some initial experimental results in Sec. II, assuming only magnetometer data are sniffed to infer Apps. More results are presented here by considering different number of Apps and different MAS approaches. In the following discussion, "Magnetic Model" denotes the baseline method where only the magnetometer data are used for training and testing; likewise, "Orientation Model" means the orientation data are used for training and testing; "+ Motion" indicates motion sensor data are also employed by following the procedure presented in Fig. 7.

We have also experimented on various devices. "Single Device" denotes the experimental setting where training and testing are carried out based on the data from a single device; "Cross Device" indicates the experiments conducted in a way that training is based on the data from a device, while testing is on a different device but of the same type (e.g., both devices are iPhone 7 Plus or both are Samsung Galaxy 7); "Cross Model" shows the results where training is based on the data from a device, but testing is on a different device of the different type; finally, "Cross Model Mix" means the setting where we mix data from different devices in different models for both training and testing. Note that, the sensors' sampling rates on iPhone 7 Plus and Samsung Galaxy 7 are different. So, under "Cross Model Mix", the training and testing datasets include data sampled at different rates. "+Downsampling" and "+Upsampling" are the signal processing schemes to decrease the sampling rate on iPhone 7 Plus or increase the sampling rate of Samsung Galaxy 7 to keep them consistent.

B. Experimental Results

As can be seen in Table IV, the accuracy generally decreases when more Apps are considered. When there



(a) (b) (c)

Fig. 8. (a) Noised vs. original magnetic data. (b) E-compass data based on noised and original magnetometer, where the noise is at the level of 8 μT. The resulting error of e-compass reading is only 0.2° in average. (c) MAS accuracy under different noise amplitude.

are more Apps, their feature distances become shorter, thus resulting in higher errors in CNN classification. For a given number of Apps, the accuracy is the lowest under "Magnetic Model (Cross Model)". This is because different smartphone models (especially different manufacturers) often use different types of magnetometers. Therefore, if we train the CNNs based on data from iPhone 7 Plus, but test them on Samsung Galaxy 7, the performance is naturally low. However, if training and testing are both based on mixed data from different device models, the performance degradation becomes negligible (see "Magnetic Model (Cross Model Mix)"). It is straightforward for an attacker to collect data from various popular phones for training purpose. In addition, "Upsampling" and "Downsampling" do not significantly improve the performance. The CNN model is not sensitive to the variance of sampling rate.

Comparing "Magnetic Model (Cross Model Mix)" and "Orientation Model (Cross Model Mix)", the latter's accuracy is only lower than the former by $4-6\%$. As discussed in Sec. II, orientation is highly correlated with magnetometer data. The use of orientation makes the attack much easier, given that it can be obtained by a Javascript embedded in the attacker's webpage.

When motion sensor-assisted MAS is employed, the accuracy becomes as high as about 0.98, and the difference between magnetometer and orientation data are fading away. This is because the difference in their accuracy does not affect the majority vote.

In addition, large electronic devices, for example, refrigerators, may generate interference on the magnetic field. To this end, we have carried out experiments when the smartphone is placed at different distances to a refrigerator. As shown in Table V, the impact is insignificant. This is because the interference is a constant and thus cancelled out after normalization.

V. Defense Mechanism

In this section, we discuss viable methods to mitigate the MAS attack. The first approach is to restrict the permission to access magnetic, orientation and motion sensors. With this method, a system notification will be popped up when an APP or Javascript requests access to those sensor data, alerting the users. Unfortunately, despite a potential threat, users may still obliviously permit such access.

TABLE V
Accuracy under Magnetic Interferences.

Distance to Refrigerator (cm)	25	50	100
Magnetic Model (Cross Model Mix) + Motion	0.9721	0.9817	0.9769
Orientation Model (Cross Model Mix) + Motion	0.9768	0.9761	0.9782

A more transparent method is noise injection that perturbs magnetic sensor output. Since the change of magnetic field caused by LED is relatively small, a minor Gaussian noises can be introduced into the magnetometer or orientation data to mitigate the attack. An example of such noise is illustrated in Fig. 8(a). It has minimum impact on normal applications. For example, the e-compass uses magnetometer data to determine how many degrees the phone's front deviates from the true North. Its results based on noised and original magnetometer signals are shown in Fig. 8(b), with the mean error of only 0.2°. For most applications, such small errors do not substantially affect their functionality.

On the other hand, the noise will significantly affect the accuracy of MAS. For instance, we generate a Gaussian noises at different levels (from $2-10$ μT) and observe their impact on MAS accuracy. As shown in Figure. 8(c), the accuracy of the magnetic model drops to 15% when the average amplitude of noise is 10 μT. Similarly the accuracy of orientation model degrades significantly with the increase of noise level. Under the motion sensors-assisted MAS which exploits motion sensor data for clustering, it is even more interesting to observe the sharp accuracy decrease when the noise level exceeds 5 μT. In this case, the majority vote is likely wrong, thus the entire cluster is classified incorrectly.

VI. Conclusion

We have discovered a new side-channel attack, where the attacker can sniff mobile Apps by analyzing magnetometer or orientation data along with motion sensor data using deep learning techniques. We have demonstrated the attack on both iPhone 7 Plus and Samsung Galaxy 7. Our experiments have shown that its accuracy is as high as 98%. We have further proposed a noise injection scheme to effectively mitigate such attacks.

REFERENCES

[1] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My Smartphone Knows What You Print: Exploring Smartphone-based Side-channel Attacks Against 3d Printers," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 895–907, 2016.

[2] A. Hojjati, A. Adhikari, K. Struckmann, E. Chou, T. N. T. Nguyen, K. Madan, M. S. Winslett, C. A. Gunter, and W. P. King, "Leave Your Phone at the Door: Side Channels that Reveal Factory Floor Secrets," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 883–894, 2016.

[3] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "TouchSignatures: Identification of User Touch Actions and PINs Based on Mobile Sensor Data via JavaScript," *Journal of Information Security and Applications (JISA)*, vol. 26, pp. 23–38, 2016.

[4] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir, "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, pp. 397–413, 2016.

[5] A. Das, N. Borisov, and M. Caesar, "Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses," in *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2016.

[6] LED Color Guide. http://www.lumex.com/article/led-color-guide.

[7] [Online] Available at:. https://www.mathworks.com/help/wavelet/ref/wden.html.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[11] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors," in *Proceedings of International Conference on Mobile Computing, Applications and Services (MobiCASE)*, pp. 197–205, 2014.

[12] S. Ha and S. Choi, "Convolutional Neural Networks for Human Activity Recognition using Multiple Accelerometer and Gyroscope Sensors," in *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, pp. 381–388, 2016.

[13] Smartphone Market Share. http://www.prnewswire.com/news-releases/comscore-reports-april-2017-us-smartphone-subscriber-market-share-300471167.html.

[14] [Online] Available at:. https://www.tensorflow.org.

[15] C.-C. Chang and C.-J. Lin, "LIBSVM – A Library for Support Vector Machines," *Proceedings of ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[16] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Proceedings of European Conference on Computer Vision (ECCV)*, pp. 818–833, 2014.

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

[18] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "Stealing PINs via Mobile Sensors: Actual Risk versus User Perception," *International Journal of Information Security*, pp. 1–23, 2016.