Open DNN Box by Power Side-Channel Attack

Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, *Member, IEEE*, Zhefu Wu, Qi Xuan, *Member, IEEE*, and Xiaoniu Yang

***Abstract*—Deep neural networks are becoming popular and important assets of many AI companies. However, recent studies indicate that they are also vulnerable to adversarial attacks. Adversarial attacks can be either white-box or black-box. The white-box attacks assume full knowledge of the models while the black-box ones assume none. In general, revealing more internal information can enable much more powerful and efficient attacks. However, in most real-world applications, the internal information of embedded AI devices is unavailable. Therefore, in this brief, we propose a side-channel information based technique to reveal the internal information of black-box models. Specifically, we have made the following contributions: (1) different from previous works, we use side-channel information to reveal internal network architecture in embedded devices; (2) we construct models for internal parameter estimation that no research has been reached yet; and (3) we validate our methods on real-world devices and applications. The experimental results show that our method can achieve 96.50% accuracy on average. Such results suggest that we should pay strong attention to the security problem of many AI devices, and further propose corresponding defensive strategies in the future.**

***Index Terms*—Deep learning, machine learning, model identification, side-channel attack, adversarial attacks.**

I. Introduction

RECENTLY, deep neutral networks (DNNs) have been the focus of research and widely used in many artificial intelligence (AI) related areas. Many DNNs are deployed and implemented on embedded devices, e.g., robots, self-driving cars, and smartphones etc., or to solve specific industrial tasks like pearl classification [1], [2], fault diagnosis [3], network control [4] and soft sensors [5]. Recently, a number of studies found that networks or systems [6], [7] are vulnerable to certain attacks [8]–[11]. On the other hand, the leakage of DNN structure may introduce new security problem. With the miniaturization of DNNs and the development of AI chips [12], DNNs on embedded hardware are becoming increasingly common and meanwhile, vulnerable to various attacks. In this brief, we demonstrate, for the first time, a side-channel attack (SCA) based technique to the embedded DNN devices which has the potential to reveal critical internal information of DNNs.

DNNs are vulnerable to adversarial attacks in the form of tiny perturbations [13]. In other words, by adding small controllable noises to the input, we can mislead the network to generate incorrect results. This poses a security problem for the application of DNN. The mainstream attacking models include black-box and white-box attacks [14]. The black-box attack assumes no prior knowledge of the neural network model. While the white-box attack relies upon the complete structure information, including both network architecture and parameter values. Apparently, the white-box attack is much more powerful and resilient than the black-box model at the cost of feasibility, i.e., most of the embedded devices are considered as black-box. To leverage this tradeoff and improve the attack performance, it is required to obtain additional neural network information for the embedded hardware.

SCA is a powerful tool to obtain hardware information. It takes advantage of side-channel signals, such as power consumption, computing time, and electromagnetic radiation etc., to reveal hidden information inside the embedded hardware [15]. Since during the computation process of the DNNs, the side-channel information shows strong correlations to the network structure and its parameters, we envision that SCA can be used for embedded AI devices and reveal their network architectures and even the corresponding parameters. Note that, as a Chinese aphorism says "*the law of defending a city is born from siege; if you want to defend well, you must attack well*", we hope this brief of SCA on DNNs could provide useful insights for researchers to propose defensive strategies in the future to better protect AI devices.

In this brief, we propose a SCA based technique to explore the network structure and parameter properties, which might be used for subsequent attacks. Our technique is based on the assumption that the AI device employs existent architectures and pre-trained parameters, which is appropriate for many real-world applications [16]. Specifically, we have made the following contributions.

1) We employ the SCA method to identify the DNN structures in embedded devices.

Manuscript received November 20, 2019; revised January 2, 2020; accepted January 25, 2020. Date of publication February 17, 2020; date of current version November 4, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61973273 and Grant 61873241, and in part by the Natural Science Foundation of Zhejiang Province under Grant LY18F030021, Grant LR19F030001, and Grant LY19F020025. This brief was recommended by Associate Editor J. Wu. *(Corresponding authors: Zhefu Wu; Qi Xuan.)*

Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Haiyang Hao, Jinyin Chen, and Qi Xuan are with the Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: xuanqi@zjut.edu.cn).

Zebin Fang and Zhefu Wu are with the College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: wzf@zjut.edu.cn).

Yi Liu is with the Institute of Process Equipment and Control Engineering, Zhejiang University of Technology, Hangzhou 310023, China.

Xiaoniu Yang is with the Science and Technology on Communication Information Security Control Laboratory, Jiaxing 314033, China.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2020.2973007

2) We use SCA to estimate the sparsity of DNN parameters. Based on the sparsity features, we derive the pre-trained parameter values.
3) We validate the effectiveness of our techniques on a real-world embedded platform.

To perform SCA, we use the power reading measurement during DNN processing. The experiment results show that our technique can achieve more than 96.5% attacking success rate.

II. Preliminary

The mainstream DNN architectures typically share some common and critical components, e.g., convolutional layers and fully connected layers etc. Moreover, parameters, or neuron weights and biases, define a DNN model. Typically, a DNN model is first trained using back propagation. During the training phase, the parameters are continuously updated. Then in the inferring phase, the parameters combined are used to perform various classification operations. Training a DNN from scratch requires numerous computation resources. Therefore, in real-world applications, people usually derive the DNN based on pre-trained parameters on existent models.

Another problem for embedded AI applications is that the computational resources on embedded platforms are very limited. To address this problem, various parameter pruning techniques were proposed [17]. Parameter pruning can cause parameter sparsity, which is defined as the proportion of zero-valued parameters. Obviously, for different DNN models with the same architecture, power consumptions can vary significantly because of different parameter sparsity. In that case, other than the DNN architecture, it is possible that the device power traces can be used to reveal the actual weights of a large portion of pre-trained neurons.

III. Side Channel Power Based Attack

In this section, we present our technique in detail. Specifically, we first develop power consumption theories and models on various DNN layers. Then, we discuss the impact of parameter sparsity on DNN power cosumptions. Finally, we describe our SCA based technique.

A. DNN Power Models

First, we build power computation models for each layer of neural network layer.

1) Convolutional Layer: Convolutional layer typically consists of many filters and is used to extract features at different level of abstractions. Specifically, the convolutional operation can be described by

$$a_{i,j} = \sum_{d=0}^{C-1}\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} \omega_{m,n}x_{i+m,j+n} + \omega_b, \tag{1}$$

where $a_{i,j}$ represents the output of a single convolutional filter, $x_{i,j}$ represents the input value, C represents the number of input channels, F represents the filter kernel size, and $\omega_{m,n}$ and ω_b represent the parameter weights and bias, respectively. Therefore, according to Eq. (1), the total operation number of a single convolutional filter is calculated by

$$\begin{cases} N_{mul} = C \times F \times F \\ N_{add} = C \times F \times F, \end{cases} \tag{2}$$

where N_{mul} is the number of multiplication operations and N_{add} is the number of addition operations. Assuming that the input size is $C \times L \times W$, the filter stride is S, and the number of convolutional kernels is N, the total operation number of the convolutional layers is calculated by

$$\begin{aligned} N_{cov} &= f_{cov}(C, L, W, S, N, F) \\ &= \begin{cases} N_{mul} = (\frac{L}{S} \times \frac{W}{S}) \times (N \times C \times F \times F) \\ N_{add} = (\frac{L}{S} \times \frac{W}{S}) \times (N \times C \times F \times F). \end{cases} \end{aligned} \tag{3}$$

Thus, the power consumption of a single convolutional layer can be derived as

$$P_{conv}(C, L, W, S, N, F) = \frac{p_m LWNCF^2}{S^2} + \frac{p_a LWNCF^2}{S^2}, \tag{4}$$

where p_m and p_a are the average multiplication and addition operation power consumptions, respectively, of the device. It is observed that the power consumption of the convolution layer is strongly correlated to the input and kernel sizes, i.e., the convolution layer architectures.

2) Pooling Layer: To reduce the growing feature dimensions without hurting the performance, there is typically a pooling layer between two consecutive convolutional layers. Without loss of generalization, we employ the maximum pooling method here. Thus, the total number of operations for the pooling layer is

$$N_{pl} = f_{pl}(C, L, W, S, F) = C \times \frac{L}{S} \times \frac{W}{S} \times F \times F. \tag{5}$$

where (C, L, W) is the input size, S is the pooling stride, and F is the pooling window size, for simplicity. Therefore, the power consumption of a pooling layer is calculated by

$$P_{pl}(C, L, W, S, F) = \frac{p_c CLWF^2}{S^2}, \tag{6}$$

where p_c represents the average comparison operation power consumption of the device. Eq. (6) indicates that the pooling layer power consumption is also largely determined by the DNN architecture.

3) Fully Connected Layer: After the convolution layer and pooling layer, DNNs typically use fully connected layers to process the extracted features. The fully connected layers consists of several layers of fully connected neurons. The operation number of a single fully connected layer can be calculated by

$$N_{fc} = f_{fc}(X, Y) = \begin{cases} N_{mul} = X \times Y \\ N_{add} = X \times Y, \end{cases} \tag{7}$$

where X is the number of input neurons and Y is the number of output neurons. Thus, the power of fully connected layer can be derived as

$$P_{fc}(X, Y) = p_m XY + p_a XY. \tag{8}$$

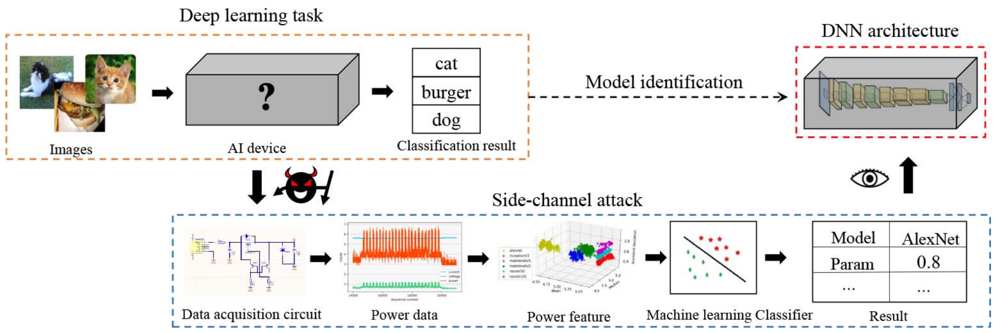

Fig. 1. The framework of our SCA based technique.

4) Activate Function: For each neuron, its value should be judged by an activation function. There are many different types of activation functions, such as relu, tanh, sigmod, and softmax, etc. The total number of operations for the activation function is calculated by

$$compute_{ac} = f_{ac}(C, L, W) = \alpha \times C \times L \times W, \tag{9}$$

where α is the operational coefficient which is determined by the specific type of the activation function. Therefore, the power consumption of the activate function can be derived as

$$P_{ac}(C, L, W) = p_{ac}\alpha CLW, \tag{10}$$

where p_{ac} is the power consumed by one operation in activation function. In general, the power consumption of the activation function is linear to the inputs and relatively small compared to other layers.

5) Overall Power Consumption: So far, we have built the power consumption models for the major components of modern DNNs. There are many other special operations. However, their computational cost is usually negligible. Therefore, in this brief, we construct our general power model based on Eq. (4), (6), (8), and (10).

It is observed that the variation of DNN architecture can have significant impact on the power consumptions. Thus, theoretically we can identify the specific architecture by analyzing and classifying the operational power traces.

B. Parameter Sparsity Model

In the previous section, we have generalized the models to identify the DNN architectures. In order to further improve the efficiency of adversarial attacks on DNN, it is desirable to derive the actual parameters, such as weights and biases, of each neuron. However, unlike the architecture, the power variation of different parameter values is very insignificant, unless it is zero. it is observed that many AI models use at least partial pre-trained parameters. Moreover, some advanced AI chips have hardware-level parameter pruning to improve the computational efficiency while maintaining the performance. Combining the pre-training and parameter pruning techniques, it is possible to further identify the actual parameter values.

It is assumed that the pre-training and pruning techniques for various models are known to the attackers. Thus, during the operation of the AI device, different pre-trained parameter sets with various pruning method can generate unique sparsity in the neurons, which can lead to differentiable power traces. Parameter pruning mainly occurs in the convolution and the fully connected layers. The power consumption of the convolution layer and that of the fully connected layer are thus changed to

$$P'_{conv}(C, L, W, S, N, F) = \lambda_1 P_{conv}(C, L, W, S, N, F), \tag{11}$$

$$P'_{fc}(X, Y) = \lambda_2 P_{fc}(X, Y), \tag{12}$$

where $\lambda_1, \lambda_2 \in (0, 1]$ are the parameter sparsity coefficients of the convolution layer and the fully connected layer, respectively. When no pruning operation is applied, the values of λ is simply set to 1. Different pre-training parameters combined with different parameter pruning techniques can lead to significant power variations. Thus, this observation can be used to identify the actual parameter values of DNN model in the AI device, which makes it even more vulnerable.

C. General Power Model

Based on the above DNN architecture and parameter power models, we build our general power model to classify the specific architecture and the corresponding parameter values. It includes training and testing phases.

The framework of our SCA on DNN models is shown in Fig. 1. We continuously collect voltage and current data while the AI device is running a model, calculate the power and power features to form a power-feature data set. We then randomly choose some data to train a classifier, and input the rest of the power-feature data into the trained classifier to get the model structure and the parameter sparsity. More specifically, for each model, we sample n pairs of current and voltage data (different DNNs have different n, for the different time they take to calculate a single image) and then calculate the average, median, and standard deviation of the power as the power features. For each model, we repeat the above sampling and data processing many times to form a power-feature data set. Finally, we design classifier D and use the data set for training. To further evaluate the parameters, we set different sparsity for each model, so the final labels include the model structure y and parameter sparsity λ.

Our SCA is summarized as Algorithm 1, where we use the classifier D, voltage array U, and current array I as the inputs, and then use D to classify and generate the outputs.

```
Algorithm 1 Side-Channel Attack (SCA)
Input: Voltage array $U = \{u_1, u_2, ..., u_n\}$ and current array
    $I = \{i_1, i_2, ..., i_n\}$ obtained by data acquisition card; The
    trained classifier $D$.
Output: Scalars $y$ and $\lambda$ representing the prediction results of
    the DNN structure and sparsity, respectively.
1: Calculate power: $P = U * I = \{p_1, p_2, ..., p_n\}$;
2: Obtain the power features:
   $p_{mea} = \sum_{i=1}^{n} p_i/n$;
   $p_{mid} = sort(P)[n/2]$;
   $p_{std} = \sqrt{\sum_{i=1}^{n}(p_i - p_{mea})^2/n}$.
3: Enter $p_{mea}, p_{mid}, p_{std}$ into the classifier $D$ to get the output
   result:
   $y, \lambda = D(p_{mea}, p_{mid}, p_{std})$.
4: Return $y, \lambda$.
```

IV. Experiments

In this section, we describe the experiments to validate the effectiveness of our techniques.

A. Experiment Setup

To extract valid power data, we use an external data acquisition card running at 400*Hz*.

In this example, we use AlexNet for image classification and continuously input 24 images (an epoch) that are randomly selected from ImageNet database. ImageNet is a large and labeled image dataset for computer vision. It contains millions of pictures and thousands of categories. All the pictures input to our models are reshaped into 224x224. The processing of the DNN requires a large amount of computing resources. Therefore, the device power increases significantly. Note that there is a start and end phase when the device runs the DNN model. Therefore, we remove those low-power phases and take the middle part of the data as the input. Due to the instantaneous nature of the power, in the process of inference, we divide per five-images' inference into a group, take all the power data, calculate their mean, median and variance as the power-feature data of the model. We implement six common DNN models on the Raspberry Pi and test them with the same set of images. After data acquisition and processing, we obtain a power-feature data set, divided into 6 categories.

B. Architecture Identification

In this brief, we use machine learning techniques to identify the DNN architectures. First, we randomly divide the power-feature data set into training set and testing set with a ratio of 4:1. Here, we simply employ the widely used SVM classifier. The results on test set are shown in Fig. 2, where the red bars is the architecture identification accuracies. The average classification accuracy reaches 96.50%. It should be noted that we do not intend to compare the performance of different machine learning algorithms, since this brief focus on proposing the overall SCA framework and SVM already achieves quite high accuracy. By using more advanced algorithms, the experimental results could be further improved.

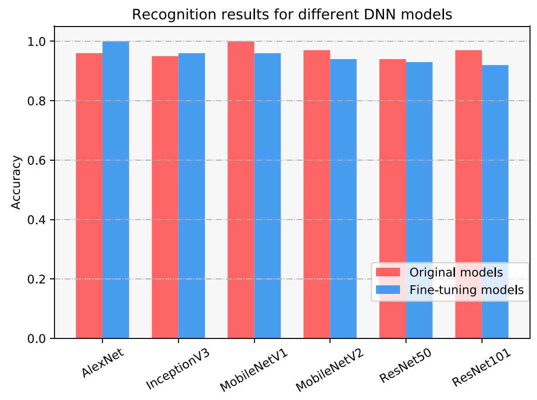

Fig. 2. Classification results for different DNN models.

In the previous experiment setup, we assume that the pre-trained DNNs are deployed exactly unchanged. However, this may not be true in the real-world applications. In general, many DNN models can be divided into two parts. The first part is for the feature extraction, such as the convolutional layers and pooling layers. The second part is for the classification, such as the fully connected layers. In real-world applications, the feature extraction part is typically with little or no change while on the other hand, the classification part is always fine-tuned based on the actual application. Fine-tuning means changes to the network structure, like removing some layers to reduce the model storage or narrowing the last fully connected later to match the number of categories in the actual data. Therefore, to further explore the performance of our technique with the fine-tuned models, we repeat the experiment with changing fully connected layer hyper-parameters. We employ some fine-tuning methods for each original architecture, including changing the number of neurons of fully connected layers, changing the number of building blocks for the ResNet. Each architecture containing four entries. The first one is the original structure and the others are the fine-tuned ones. The power traces from all the fine-tuned architectures are mixed with the original ones. Then the architecture identification process is repeated on the new data set.The classification results for fine-tuning models are still shown in Fig. 2 (blue bars). Fine-tuning can increase or decrease the overall computational costs of the model and thus, affecting the classification results. Model like AlexNet has a relatively simpler architecture. Therefore, decreasing its fully connected layers can impact significantly the computational power consumption and make it easier to distinguish from other models. While ResNet has a more complicated structure. The average performance is slightly reduced to 95.17%, which demonstrates that our SCA is quite robust even with the fine-tuning of the classification layers.

C. Parameter Evaluation

The evaluation of the model parameters is also important. For example, the traditional white-box attack requires full knowledge of the DNN model, including parameter values. However, estimating the model parameters are usually quite challenging. In this brief, we assume that pre-trained architectures and partial model parameters could be employed in

TABLE I
ACCURACY FOR DIFFERENT PARAMETER SPARSITY

Models	Parameter sparsity	Accuracy	Average
AlexNet	1.0	88%	54.25%
	0.8	36%	
	0.6	44%	
InceptionV3	1.0	94%	82.75%
	0.8	86%	
	0.6	76%	
ResNet50	1.0	100%	84.00%
	0.8	84%	
	0.6	65%	
ResNet101	1.0	100%	84.5%
	0.8	64%	
	0.6	88%	

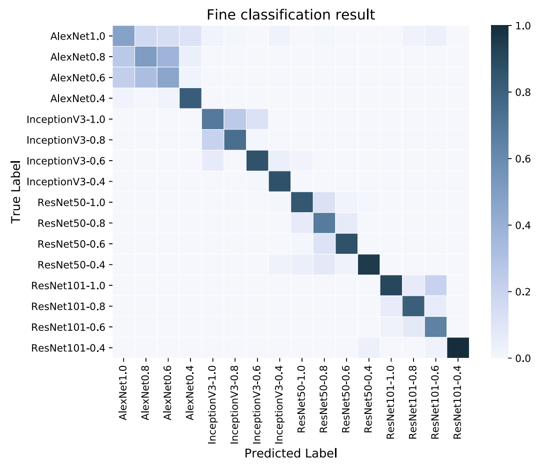

Fig. 3. The confusion matrix for the fine classification of DNN models.

certain real-world applications. Moreover, due to the techniques such as parameter tunings, different models could have varying sparsity, i.e., the ratio of zeros. We thus use the model sparsity as features to infer the model parameters.

Specifically, we employ dropout in all the convolutional and fully connected layers and set different dropout scales to derive various parameter sparsity. The dropout scale is set to 1.0, 0.8, and 0.6, respectively. In this experiment, we select four DNN models. The classification results of the same model with varying parameter sparsity are shown in Table I. The average identification rate is about 76.38%. The AlexNet is relatively hard to identify since it is simple and small. Therefore, it has less parameters and hence, less impact on power consumption. From the perspective of attack, the parameter sparsity based attack may be less efficient on AlexNet. However, in general, for the large networks, the accuracy is still well above 82%. We have performed model architecture and model parameter estimation. However, these two processes may affect each other. Therefore, we need to check the general model identification results which include both varying architecture and varying parameters.

By mixing up the architecture and parameter sparsity variations, we have a total of 16 different categories in the data set. The experimental results are shown in Fig. 3. The confusion matrix demonstrates that, even with varying parameter sparsity, we can still get the structure of a DNN model with relatively high accuracy, i.e., more than 95%. The accuracy of parameter sparsity recognition under the fine classification task is 75.88%, which is reduced a little but generally acceptable.

V. Conclusion

In this brief, we propose a side-channel attack (SCA) method to reveal the internal structure and model parameters for DNN models. We design a raspberry pi based platform to derive the power signature of embedded AI devices, and then use machine learning algorithms to identify the specific DNN architectures. Moreover, we differ the parameter sparsity to model the pre-training of DNNs. In general, our technique can identify both the architecture and model parameters with quite high accuracy, indicating that we should pay strong attention to the security problem of many AI applications.

References

[1] Q. Xuan, Z. Chen, Y. Liu, H. Huang, G. Bao, and D. Zhang, "Multiview generative adversarial network and its application in pearl classification," *IEEE Trans. Ind. Electron.*, vol. 66, no. 10, pp. 8244–8252, Oct. 2019.

[2] Q. Xuan *et al.*, "Automatic pearl classification machine based on a multistream convolutional neural network," *IEEE Trans. Ind. Electron.*, vol. 65, no. 8, pp. 6538–6547, Aug. 2018.

[3] J. Chen, Y. Yang, K. Hu, Q. Xuan, Y. Liu, and C. Yang, "Multiview transfer learning for software defect prediction," *IEEE Access*, vol. 7, pp. 8901–8916, 2019.

[4] Y. Lou, Y. He, L. Wang, and G. Chen, "Predicting network controllability robustness: A convolutional neural network approach," 2019. [Online]. Available: arXiv:1908.09471.

[5] Y. Liu, C. Yang, Z. Gao, and Y. Yao, "Ensemble deep kernel learning with application to quality prediction in industrial polymerization processes," *Chemometr. Intell. Lab. Syst.*, vol. 174, pp. 15–21, Mar. 2018.

[6] Y. Huang, J. Wu, W. Ren, K. T. Chi, and Z. Zheng, "Sequential restorations of complex networks after cascading failures," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, doi: 10.1109/TSMC.2018.2874822.

[7] Z. Chen, J. Wu, Y. Xia, and X. Zhang, "Robustness of interdependent power grids and communication networks: A complex network perspective," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 1, pp. 115–119, Jan. 2018.

[8] S.-W. Sun, Y.-L. Ma, R.-L. Li, L. Wang, and C.-Y. Xia, "Tabu search enhances network robustness under targeted attacks," *Physi. A, Stat. Mech. Appl.*, vol. 446, no. 446, pp. 82–91, Mar. 2016.

[9] S. Sun, Y. Wu, Y. Ma, L. Wang, Z. Gao, and C. Xia, "Impact of degree heterogeneity on attack vulnerability of interdependent networks," *Sci. Rep.*, vol. 6, Sep. 2016, Art. no. 32983. [Online]. Available: http://europepmc.org/articles/PMC5016735

[10] W. Yang, Z. Zheng, G. Chen, Y. Tang, and X. Wang, "Security analysis of a distributed networked system under eavesdropping attacks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, doi: 10.1109/TCSII.2019.2928558.

[11] S. Yu *et al.*, "Target defense against link-prediction-based attacks via evolutionary perturbations," *IEEE Trans. Knowl. Data Eng.*, early access, doi: 10.1109/TKDE.2019.2933833.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," 2018. [Online]. Available: arXiv:1801.04381.

[13] C. Szegedy *et al.*, "Intriguing properties of neural networks," 2013. [Online]. Available: arXiv:1312.6199.

[14] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," 2018. [Online]. Available: arXiv:1801.00553.

[15] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," in *Proc. IACR Cryptol. ePrint Archive*, vol. 2005, no. 388, 2005.

[16] A. Singla, L. Yuan, and T. Ebrahimi, "Food/non-food image classification and food categorization using pre-trained googlenet model," in *Proc. 2nd Int. Workshop Multimedia Assist. Dietary Manag.*, 2016, pp. 3–11.

[17] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning." in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2018, pp. 2425–2432.