Fake Gradient: A Security and Privacy Protection Framework for DNN-based Image Classification

Xianglong Feng
Rutgers University
Piscataway, NJ, USA
xf56@scarletmail.rutgers.edu

Yi Xie
Rutgers University
Piscataway, NJ, USA
yi.xie@rutgers.edu

Mengmei Ye
Rutgers University
Piscataway, NJ, USA
mengmei.ye@rutgers.edu

Zhongze Tang
Rutgers University
Piscataway, NJ, USA
zhongze.tang@rutgers.edu

Bo Yuan
Rutgers University
Piscataway, NJ, USA
bo.yuan@soe.rutgers.edu

Sheng Wei
Rutgers University
Piscataway, NJ, USA
sheng.wei@rutgers.edu

ABSTRACT

Deep neural networks (DNNs) have demonstrated phenomenal success in image classification applications and are widely adopted in multimedia internet of things (IoT) use cases, such as smart home systems. To compensate for the limited resources on the IoT devices, the computation-intensive image classification tasks are often offloaded to remote cloud services. However, the offloading-based image classification could pose significant security and privacy concerns to the user data and the DNN model, leading to effective adversarial attacks that compromise the classification accuracy. The existing defense methods either impact the original functionality or result in high computation or model re-training overhead. In this paper, we develop a novel defense approach, namely *Fake Gradient*, to protect the privacy of the data and defend against adversarial attacks based on encryption of the output. *Fake Gradient* can hide the real output information by generating fake classes and further mislead the adversarial perturbation generation based on fake gradient knowledge, which helps maintain a high classification accuracy on the perturbed data. Our evaluations using ImageNet and 7 popular DNN models indicate that *Fake Gradient* is effective in protecting the privacy and defending against adversarial attacks targeting image classification applications.

CCS CONCEPTS

• **Information systems** → **Multimedia information systems**; • **Security and privacy** → **Domain-specific security and privacy architectures**;

KEYWORDS

Image classification; deep neural network; adversarial attack

MM '21, October 20–24, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475685

ACM Reference Format:
Xianglong Feng, Yi Xie, Mengmei Ye, Zhongze Tang, Bo Yuan, and Sheng Wei. 2021. Fake Gradient: A Security and Privacy Protection Framework for DNN-based Image Classification. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475685

1 INTRODUCTION

Deep neural networks (DNNs) have been widely adopted in many multimedia applications, such as image classification [30], natural language processing [15], and medical image analysis [3], and have demonstrated great success in the multimedia community in the recent years. Among them, DNN-based image classification [30] has become one of the most popular and fundamental building blocks to support many advanced use cases, with a new trend of being rapidly adopted by mobile and internet of things (IoT) platforms. For example, IoT-based smart home devices have utilized DNN-based image classification and face recognition to enable the important intruder detection feature [20, 24].

The major challenge in deploying and executing DNN-based image classification on IoT platforms is the big gap between the computation-intensive DNN operations and the limited computation and power resources on the battery-driven mobile devices. To address this challenge, the IoT-based image classification applications often offload the computation-intensive DNN execution to a cloud or edge server [35], which has abundant computation resources to complete the classification task with high efficiency.

However, such offloading-based DNN execution could pose significant security and privacy concerns to the user data and the DNN model which, if exposed to adversaries, could lead to breach of confidentiality/integrity in the input data and/or the inference output. One notable example is the adversarial attacks [9] against DNN models, which intend to add small, human-imperceptible perturbations to the user input image to compromise the correctness of the inference results. Many recent studies have demonstrated that such adversarial attacks can be effectively achieved by adversaries who gained access to the user data and DNN model [9, 16, 19, 29, 36], a feasible scenario in the case of untrusted DNN offloading.

To address the aforementioned security and privacy issues, we aim to develop an effective defense mechanism achieving the following goals of privacy and security protections:

- ***G1 (Privacy): Hiding classification results.*** Although the cloud service providers can execute the DNN model based on the standard machine learning computing rules and gain access to the output vectors, they should not be able to decode the correct classification results from the output vector.
- ***G2 (Security): Resilient to the adversarial attack.*** If the attacker adopts the adversarial attacks and attempts to mislead the DNN-based classification, such an attempt should fail in that the perturbed image should still be classified into the correct class by the DNN model.

Also, considering the system and application requirements of the image classification operations, the defense mechanism should satisfy the following functionality and cost requirements:

- ***R1 (Functionality): Maintaining the original functionality of image classification.*** The defense mechanism, if requiring enhancement to the DNN model or the input data, should not worsen the original classification accuracy of the DNN model.
- ***R2 (Cost): Minimal processing overhead.*** The defense mechanism should minimize its processing overhead to avoid impacting the original efficiency (e.g., real-time) requirement of the image classification application.

Targeting ***G1***, researchers proposed to use encryption for privacy protection [4, 6, 33]. However, the encryption-based approaches have two limitations. First, the encrypted data does not directly work with the original weights of the model and, therefore, they often require training a new DNN model based on the encrypted data. Also, the classification accuracy by the re-trained model may not be as high as that of the original model, which does not meet ***R1***. Second, these methods require the IoT devices to encrypt the data for offloading, which introduces nontrivial processing overhead and would violate ***R2***.

Other works focusing on ***G2*** aim to defend against the adversarial attacks, which can be categorized into two types, namely adversarial training [8, 14, 17, 26, 28] and perturbation removal [27, 34]. The adversarial training method modifies the model and requires re-training with adversarial samples and, consequently, it may impact the original functionality of image classification. The perturbation removal method could achieve better defense effectiveness, however, it could also compromise the original functionality of the DNN model. As a result, it is challenging for both approaches to fulfill ***R1***.

Focusing on both ***G1*** and ***G2***, we propose an efficient protection framework, namely *Fake Gradient*, which can hide the real classification results by misleading the attacker to access the fake outputs (i.e., ***G1***) and defend against the adversarial attacks with fake gradient in the DNN model (i.e., ***G2***). To achieve this goal, we add new output nodes (i.e., the fake outputs) to the original output layer (i.e., the last fully connected layer) of the DNN model. The new nodes are designed to have higher values to ensure that the calculations in the DNN would mislead the attacker to the fake outputs. Also, they are designed to generate a fake gradient for each fake output to make the adversarial attack generate less effective perturbations. In addition, we design a random key-based encryption method, with which the IoT device could deeply mix the real and fake outputs and recover the correct outputs in a secure manner.

We evaluate the proposed *Fake Gradient* approach using the ImageNet dataset [7] with 7 popular convolutional neural networks (CNNs). The results reveal that the inference intended by the attacker would fall in the fake outputs. Also, input images that belong to the same class can be inferred as different classes based on the fake outputs. Furthermore, the intermediate results show that the fake gradient of each fake output can mislead the adversarial attack to generate less effective perturbations and thus defeat the attacker's attempts.

To summarize, our proposed *Fake Gradient* framework presents novel contributions with several advantages in meeting the aforementioned security/privacy goals (***G1*** and ***G2***) and requirements (***R1*** and ***R2***) for IoT-based image classification tasks. First, the proposed encryption approach is based on the standard deep learning computing process, which can be seamlessly integrated with arbitrary platforms and DNN libraries. Second, the fake outputs with fake gradients could effectively protect the user privacy and defend against the adversarial attacks. Last but not least, to the best of our knowledge, this is the first work to encrypt the DNN models based on the fully connected layer. The modifications to the model introduce little computation overhead to the inference process, as the majority of the DNN is the convolution layers. The proposed approach does not compromise the original functionality of the model and thus does not require re-training, which successfully meets the ***R1*** and ***R2*** requirements while achieving the ***G1*** and ***G2*** goals in the image classification applications.

2 BACKGROUND: ATTACKS ON DNN-BASED IMAGE CLASSIFICATION

In this work, we focus on the multimedia security and privacy issues in the IoT-based smart home systems, such as the image classification-based intruder detection. In this application, as shown in Figure 1, the smart home device (e.g., a smart camera) intends to run a DNN-based face image classification to inspect if a visitor to the house is an unrecognized intruder. To avoid deploying and executing the large DNN-based image classification model on the resource constrained IoT device, the system is designed to offload the model and the collected input images to the cloud for image classification.

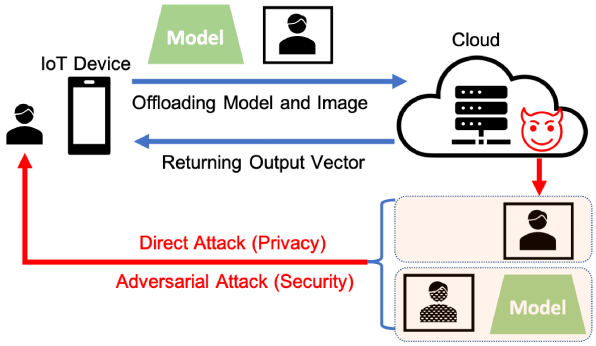

Figure 1: Attack scenario in DNN-based image classification.

However, the cloud is typically untrusted due to security and privacy considerations from the user's perspective. In this work,

we mainly target two types of attacks, which can be issued by attackers who gain access to the images and models. In the first attack method, the attacker attempts to gain access to and thus breach the privacy of the input image (e.g., the face image of the home owner) by interpreting the classification results, which can be further leveraged to bypass the access control module in the smart home system, namely *direct attack* [1, 13]. The second attack, namely *adversarial attack* [9, 16, 19, 29, 36], is to add perturbations to the input image, which are less noticeable to humans but would lead the original DNN model to generate erroneous classification results (e.g., misidentify the intruder as the home owner).

3 RELATED WORK

DNN security has drawn a great deal of attention in the community recently, and there have been many research efforts on addressing the aforementioned attacks. Several works [4, 6, 33] propose to use encryption to protect the user privacy and thus defend against the *direct attacks*. Other works [8, 14, 17, 21, 22, 26–28, 34] focus on the resilience of the DNN models to the *adversarial attacks*. However, none of the existing approaches can effectively address the privacy and security goals and requirements targeted by this paper (i.e., the G1/G2 goals and the R1/R2 requirements).

3.1 Defense against Direct Attack

Targeting the *direct attack*, several works have been conducted to encrypt the offloaded input data. In [33], the proposed approach transforms the real-valued features into complex-valued ones, in which the input is hidden in a randomized phase of the transformed features. The knowledge of the phase acts like a key, which is required to recover the output from the processing results. Several other works [4, 6] leverage homomorphic encryption to encrypt the input data, which, although demonstrating good effectiveness of protection, have several limitations. First, they introduce significant processing overhead to the IoT device due to the required encryption. Second, the encrypted data cannot directly work with the original DNN model, and thus they require retraining the model.

3.2 Defense against Adversarial Attack

Many research works [8, 14, 17, 21, 22, 26–28, 34] have targeted the *adversarial attacks*, which can be divided into two categories. (1) *Improving the DNN model*: Following the idea of "learning", some works propose to let the model learn from the perturbed data and still yield correct classification results under adversarial perturbations [8, 14, 17, 26, 28], which are also known as "adversarial training". Others [21, 22] exploit the notion of "distillation" [11] to make DNNs resilient to adversarial attacks by transferring the knowledge of a more complex network to a smaller network. These methods all require re-training the model, which introduce significant processing and deployment overhead. (2) *Perturbation removal*: The other line of work focuses on removing the adversarial perturbations from the input images. Xu et al. [34] propose to reduce the complexity of the data representation so that the adversarial perturbations would disappear due to low sensitivity. Shaham et al. [27] investigate various defense mechanisms, such as PCA, JPEG compression and soft thresholding, which are conducted before feeding the image into the model. The results show that JPEG compression performs better than the other defence mechanisms in most of the scenarios. Although these techniques work well in preventing adversarial attacks, they have the collateral effect of worsening the accuracy of the model on true examples.

4 PROPOSED APPROACH: FAKE GRADIENT

4.1 Challenges

In the deployment of DNN frameworks, the model is typically visible to the computing platform. On one hand, to use the platform, the user needs to provide the weights and the structure of the model. Therefore, the platform knows the value of the model parameters. On the other hand, the computation of the DNN model is very standard for each layer (e.g., convolution, fully connected, and dropout) and is supported by the standard libraries from third parties (e.g., TensorFlow [2] and PyTorch [23]). All of these make the original DNN model parameters known by the cloud, which poses significant challenges for the security/privacy protection mechanisms. Also, since the DNN model is trained based on finding the gradient to minimize the loss, the *adversarial attack*, which is also based on the gradient, is often effective to fool the neural network [5, 32].

To overcome the limitations of the existing methods, we propose a *Fake Gradient* method, which encrypts the target DNN model in the last fully connected layer based on the standard neural network computation. As a result, the attacker would be misled to the fake output and would obtain the fake gradient, which eventually achieves the security and privacy goals.

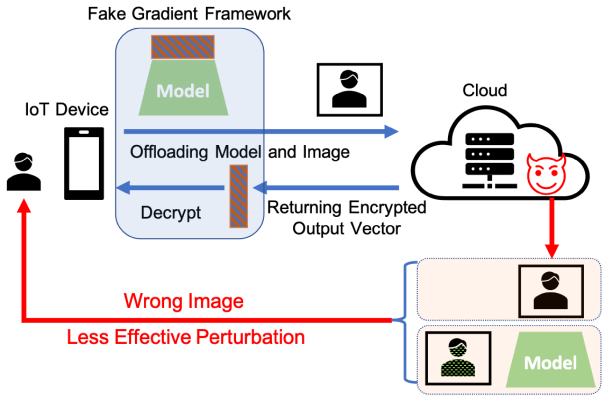

Figure 2: *Fake Gradient* system architecture.

4.2 *Fake Gradient* Architecture

Figure 2 shows the system architecture and workflow of the proposed *Fake Gradient* approach in the image classification system. *Fake Gradient* modifies the fully connected layer of the original DNN model and deploys it to the cloud for execution. At runtime, the IoT device captures and offloads the input images to the cloud for real-time image classification, and the cloud returns the output vectors of the DNN model to the IoT device. Given the modified

DNN model, the returned output vectors are encrypted and, therefore, the attackers who do not possess the key would obtain the fake inference results with fake gradient information. As a result, the attackers cannot uncover the sensitive input image or generate effective adversarial perturbations to mislead the DNN model. However, the legitimate users can first decrypt the output vectors using the key and, eventually, obtain the correct classification results.

4.3 Design Principles of *Fake Gradient*

Based on the previous discussions, the core of the *Fake Gradient* framework is the design of the new weight connections in the enhanced DNN model. In particular, we define three concrete design principles for *Fake Gradient* following the goals and requirements discussed in Section 1.

- ***Adding fake output without compromising the original functionality of the DNN model:*** Adding new neurons to the DNN model would change the structure of the model and introduce new weights, which would eventually change the original inference results. As discussed in ***R1***, we must maintain the original DNN inference accuracy. Otherwise, we need to retrain the model to maintain its original functionality, which would introduce additional computation overhead and conflict with ***R2***. Therefore, the first design principle of introducing new weights in *Fake Gradient* is to maintain the model's functionality without re-training.
- ***Misleading the direct and adversarial attacks:*** To defend against the *direct attack*, we aim to mislead the final prediction results by ensuring that the forward propagation in the DNN model would fall into the fake output classes with high probability. Also, to defend against the *adversarial attack*, we must further minimize the impact of the generated perturbations and make them insensitive to the DNN model, as they would degrade the classification accuracy even if being generated based on the fake classes.
- ***Key-based encryption/decryption of the outputs:*** For the deployment of *Fake Gradient*, we aim to design a key-based encryption and decryption mechanism to generate and recover from the *Fake Gradient*-based outputs in the offloading-based image classification workflow, which can be customized to individual users and applications to further enhance the security and privacy.

Considering all the above design principles, we divide the model modification process of *Fake Gradient* into two components, namely the design of the new weights and the output encryption, as shown in the upper diagram of Figure 3 and discussed in details in Section 4.4.1 and Section 4.5, respectively. The model modification process is then integrated into the end-to-end workflow of *Fake Gradient*, as shown in the bottom diagram of Figure 3, followed by the output decryption step for the legitimate user to recover the real outputs.

4.4 Design of the New Weights

4.4.1 Fake output. At the very first step of our *Fake Gradient*, we only consider how to generate the fake output without degrading the performance of the DNN model. Here we define x as the input image and y as the corresponding true label. The DNN model contains several convolution layers $Con(\cdot)$, followed by the last fully connected layer $Fc(\cdot)$. The computation of the target model $\phi(x)$ can be represented as follows:

$$\phi(x) = Fc(Con(x)) \tag{1}$$

Assuming the outputs from the convolution layers $Con(\cdot)$ are V_c, the original weight matrix of the fully connected layer is W_o, and the original bias of the fully connected layer is b_o, the original output vector can be calculated as $f_o = V_c \bigotimes W_o + b_o$. Then, the final output Y can be formulated as follows:.

$$Y = argmax(f_o) = argmax(V_c \bigotimes W_o + b_o) \tag{2}$$

To add new neurons to the output layer, we increase the number of weights and the bias of the original fully connected layer. Here we set the new weights as $[W_o, W_F]$ and the bias as $[b_o, b_F]$. Therefore, the new output vector based on the modified fully connected layer is $V_c \bigotimes [W_o, W_F] + [b_o, b_F]$. This result can be further written as $\left[V_c \bigotimes W_o + b_o, V_c \bigotimes W_F + b_F\right]$, where $V_c \bigotimes W_o + b_o$ is the original output Y, and $V_c \bigotimes W_F + b_F$ is the fake output y. Therefore, the new output vector contains both the original and fake classes, which meets the first design principle discussed in Section 4.3.

By now the two variables we need to address are the new weight W_F and the new bias b_F. We note that in the DNN calculation, as shown in Equation (2), the bias is fixed and unrelated to the input images. This means that we cannot modify the bias to generate the fake gradient. Therefore, we first set $b_F \leftarrow b_0$. Then, the next step is to fulfill the second design principle (discussed in Section 4.3) by devising the new weight matrix to make the inference result fall into the new output nodes. Based on the computation rules of neural networks, the inference result is the output node with the highest value. Given the modified output (i.e., $\left[V_c \bigotimes W_o, V_c \bigotimes W_F\right]$, ignoring the bias b since $b_F = b_0$) based on the new weight matrix $[W_o, W_F]$ of the modified fully connected layer, the goal is to achieve $max(V_c \bigotimes W_o) < max(V_c \bigotimes W_F)$ so that $Y \neq y$. Here we can set a scaling factor $S_a > 1$ and set $W_F = S_a \cdot W_o$. Therefore, the new weights are $[W_o, S_a \cdot W_o]$, based on which we ensure that the inference output would fall into the new output nodes, since $max(V_c \bigotimes W_o) < max(V_c \bigotimes S_a \cdot W_o)$.

4.4.2 Fake gradient. Setting the new weights as $[W_o, S_a \cdot W_o]$ ensures that the inference result would fall into the new output nodes. However, the new inference result is still based on original weights W_o and can be used by the adversarial attack to generate effective perturbations. Considering the ***G2*** goal, our next step is to design a fake gradient to mislead the adversarial attack.

The intuitive idea is to generate the new weight W_F randomly so that, based on the output, the adversarial attack would fail to obtain the true gradient. However, the adversarial attack algorithm would still be able to generate the perturbations to the images, and the perturbations could still impact the original classification accuracy to some extent. Note that the goal of the adversarial attack is to generate the minimum perturbation r that can greatly change the inference result of $x + r$ [19], as described by Equation (3):

$$minimize \, \|r\|_2, \text{ such that } \phi(x) \neq \phi(x + r) \tag{3}$$

To achieve effective adversarial perturbations, the adversarial attack algorithm first calculates the gradient $\triangledown\phi(x)$, based on which it further generates the minimum r to change the final inference result, i.e., $\phi(x) \neq \phi(x + r)$.

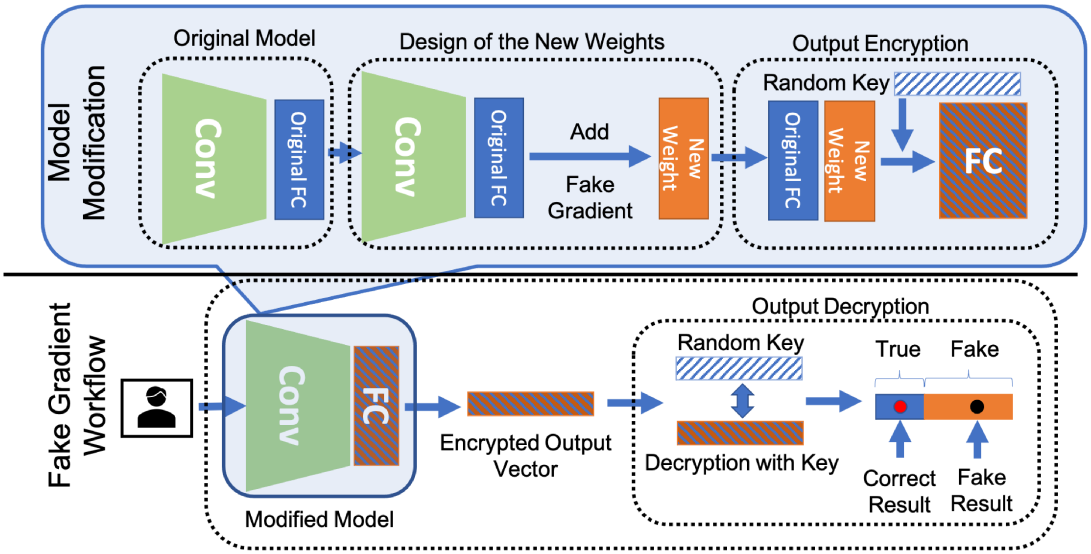

Figure 3: Workflow of the *Fake Gradient* framework.

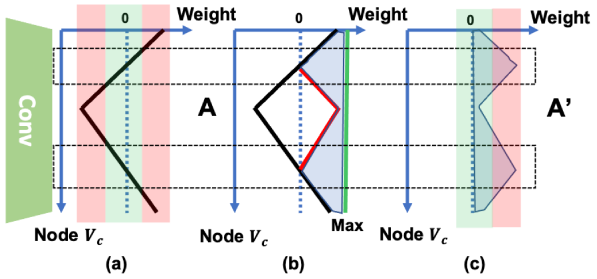

Figure 4: Generating new weights based on *Fake Gradient*.

To defeat the adversarial attack, our proposed *Fake Gradient* would perform the opposite, which aims to lead the adversarial attack to generate more perturbations but barely change the inference result. The flow of generating the new weights with misleading gradient is shown in Figure 4, where the vertical axis represents the output of the convolution layer (i.e., V_c), and the horizontal axis represents the values of the weights connected to the node A in the next layer (i.e., the output vector f). In Figure 4(a), we label the areas with red and green, which represent the weights with high and low absolute values, respectively. In other words, the nodes of V_c with weights in the green zone have less influence on obtaining the inference result of A than the nodes with weights in the red zone. Assuming that A' is the fake output, we aim to design the new weights for A' with fake gradient. In other words, the nodes that have weights in the green zone for A should have weights in the red zone for A', as shown in Figure 4(c).

The details of how to generate the weights for the new nodes are described in Algorithm 1. Once we have the DNN model M, we can abstract the fully connected layer L_{fc} with its original weights and bias (i.e., W_o, b_o). Then, we enlarge the fully connected layer by doubling the weights and bias (i.e., $[W_o, W_F]$, $[b_o, b_F]$). For

Algorithm 1: Fake Gradient: New outputs & new weights

Input: The original DNN model M, scaling factor S_a.
Output: The modified DNN model M'.
Get the fully connected layer $L_{fc} < W_o, b_o, f_o > \leftarrow M$;
Construct the new fully connected layer $L'_{fc} < [W_o, W_F], [b_o, b_F], [f_o, f_F] >$;
$b_F \leftarrow b_o$;
$W_{max} \leftarrow Max(W_o)$;
for *Index of row and column i,j in* W_o **do**
  $W_F[i][j] = (W_{Max} - |W_o[i][j]|) \cdot S_a \cdot sign(W_o[i][j])$;
end
Generate the new model $M' \leftarrow L'_{fc}$;

the bias, we just directly copy the original bias to the new bias. For the weights, we first obtain the the maximum value of the weights W_{Max} and, then, subtract the absolute value for one weight $W_o[i][j]$ from W_{Max}, which is shown in Figure 4(b). The obtained value is modified by the scaling factor S_a and applied the same sign as the original weight. Finally, it is assigned to the fake weight $W_F[i][j]$. We follow this procedure to complete the entire W_F.

4.5 Output Encryption

The current method of generating the new weight W' combines the original weight W_o and the generated fake weight W_F directly, which is easy to be recognized by the space pattern. To improve it, the advanced method should mix the two weight matrices deeply so that it becomes difficult for the attacker to uncover the original outputs and the original weights. As shown in Figure 5, we change the order of the rows and columns of the weight matrices to mix them deeply. Based on the computation rule of the fully connected layer, we notice that we can change the output by changing the order of the rows; similarly, we can change the order of the input

vector to change the order of the columns in the matrix without compromising its original inference accuracy. However, since the order of the input vector is determined by the previous layers (i.e., the convolution layers), we only mix the weight matrices by changing the order of the rows.

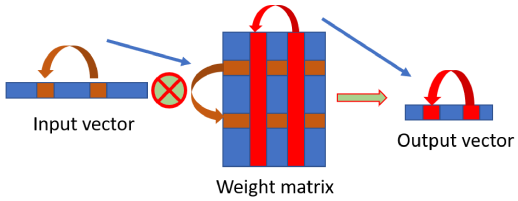

Figure 5: Rearrangement of the input/output vector by changing the row/column of the weight matrix.

4.5.1 Encryption with random key. Here we use ω_i to represent the i^{th} column of the new weight matrix W' for the fully connected layer, b_i to represent the i^{th} bias, and fc_i to represent the i^{th} output of the fully connected layer f'. Based on Equation (2), we can further calculate each output of the fully connected layer fc_i, as follows:

$$fc_i = V_c \cdot \omega_i + b_i \tag{4}$$

As shown in Equation (4) and Figure 5, we can change the order of the output nodes and that of the rows in the weight matrix following the same pattern (i.e., the order of i), which not only mixes the weight matrices but also encrypts the output. The modification pattern can be defined by a random key (i.e., K), which is a vector with the same length as the output vector of the modified DNN model. When generating K, we assign the index of each item and randomly permute all the items in K. Now each item in K contains a unique number indicating the new position of the output node.

Algorithm 2: Fake Gradient: Output encryption

1 **Input:** The modified DNN model M'.
2 **Output:** The encrypted DNN model M_{en}, random key K.
3 Get the fully connected layer $L'_{fc} < [W_o, W_F], [b_o, b_F], [f_o, f_F] > \leftarrow M'$;
4 Generate the random key K;
5 $W' \leftarrow [W_o, W_F]$;
6 $b' \leftarrow [b_o, b_F]$;
7 $f' \leftarrow [f_o, f_F]$;
8 **for** *Column index i in* W' **do**
9 $\quad W_{en}[K[i]] \leftarrow W'[i]$;
10 $\quad b_{en}[K[i]] \leftarrow b'[i]$;
11 **end**
12 Generate the new model $M_{en} \leftarrow L_{fc} < W_{en}, b_{en}, F_{en} >$;

Algorithm 2 shows the details of how we use K to encrypt the model. To generate the encrypted weight W_{en} based on the modified weight W' and the key K, we check the corresponding value in $K[i]$ and assign the i^{th} row of the weight matrix W' to the $K[i]^{th}$ row of the weight matrix W_{en}. Following this procedure, we further modify the bias b_{en} by assigning the value of the i^{th} item in the bias b' to the $K[i]^{th}$ item in the new bias b_{en}. This encryption approach also enables the user to adopt customized keys and thus generate different protected inference results for different cloud services and application scenarios, which further enhances the security and privacy.

4.5.2 Decryption with random key. As shown in the output decryption block of Figure 3, the user can decrypt and recover the original output vector using the key K upon receiving the encrypted results from the cloud. Algorithm 3 describes the output decryption algorithm. To obtain the i^{th} output value of the modified output layer f', we check the value of the $K[i]^{th}$ output value of the encrypted output vector F_{en}. In the end, we find the highest score from the first half of f', which is the original output vector f_o, to obtain the correct inference result.

Algorithm 3: Fake Gradient: Output decryption

1 **Input:** Returned output vector f_{en}, random key K.
2 **Output:** Decrypted output vector f_o.
3 **for** *Item index i in* f_{en} **do**
4 $\quad f'[i] \leftarrow f_{en}[K[i]]$;
5 **end**
6 $[f_o, f_F] \leftarrow f'$;

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

We evaluate our *Fake Gradient* approach using ImageNet [7] on 7 DNN models, including ResNet18, ResNet34 [10], DenseNet121 [12], VGG11, VGG19 [30], MobileNet [25], and GoogLeNet [31]. To maintain the generality, we obtain these models from the torchvision library [18] with pre-trained weights. The hardware platform is a Dell workstation T7910 with two Intel Xeon E5-2623 CPUs, 32GB RAM and an Nvidia GTX TITAN X GPU with CUDA 10.0.

The evaluations mainly focus on the security/privacy goals (***G1*** and ***G2***) and the requirements (***R1*** and ***R2***) for image classification tasks, which are achieved by evaluating the encryption of the output, the efficiency of defense against adversarial attacks, and the overhead. Considering that the other related works cannot address the same goals and requirements targeted by this work, we compare the defense performance with a random weight approach, in which the new weight in line 8, Algorithm 1 is generated by a Gaussian function fitting the distribution of the original weights.

5.2 Output Encryption

We first evaluate the effectiveness of output encryption in hiding the real output. We compare the model output from *Fake Gradient* (without decryption) with the original DNN output, checking whether the inference result would fall into the fake outputs. As shown in the first two rows of Table 1, all the test images with *Fake Gradient* are inferred as fake outputs, while the random weight approach can only generate fake outputs for very few test images.

Also, we evaluate whether *Fake Gradient* would compromise the original functionality of the DNN model. In this evaluation,

Table 1: The evaluation results over the four evaluation metrics comparing the *Fake Gradient* and Random Weight approaches.

Evaluation Metrics	Methods	ResNet34	ResNet18	DenseNet121	VGG19	VGG11	MobileNet	GoogLeNet
Fake Output	Random Weight	1	1	0	0	0	0	6
(%)	*Fake Gradient*	100	100	100	100	100	100	100
Original Functionality	Random Weight	100	100	100	100	100	100	100
(%)	*Fake Gradient*	100	100	100	100	100	100	100
Defence Success Rate	Random Weight	1.2	11.4	0	0	0	0	22.5
(%)	*Fake Gradient*	90.5	92.1	94.9	79.8	82.9	87.1	84.8
Perturbation	Random Weight	0.0019	0.0016	0.0017	0.0017	0.0020	0.0012	0.0017
(L2 norm)	*Fake Gradient*	0.0010	0.0009	0.0008	0.0015	0.0015	0.0012	0.0015
	Original Model	0.0018	0.0016	0.0017	0.0017	0.0020	0.0012	0.0018

Table 2: The random outputs for the same input image using *Fake Gradient* without the random key-based encryption.

Classification Results for Images of Class 814					
Image ID	198	611	921	2021	2037
Inference Result	1977	1356	1693	1693	1733
Original Classification Results For Images Classified as Class 1733					
Image ID	1753	1869	2032	2114	2220
Original Class	645	702	814	724	416

we compare the inference results generated by the original model and the modified model (after decryption), as shown in the third and fourth rows of Table 1. The results indicate that *Fake Gradient* achieves the same inference results as the original model for all the test images, which indicates that *Fake Gradient* can maintain the model's original functionality without re-training.

Furthermore, Table 2 shows the random output from *Fake Gradient* without the random key-based encryption. We observe that, in this case, even the images that belong to the same class could be classified into different fake classes, and the images that are inferred as the same fake output could originally belong to different classes. These results demonstrate the randomness of *Fake Gradient*, which is effective in misleading the attacker and protect the privacy.

5.3 Defense Against Adversarial Attacks

Figure 6 demonstrates the intermediate results of *Fake Gradient* from the sample test images based on ResNet34. The first column shows the original images. The second and third columns show the perturbations generated by *Fake Gradient* and on the original gradient, respectively, in the form of heatmap. The fourth column demonstrates the difference between the two types of gradients. The green dots represent the gradients based on the model modified by *Fake Gradient*, the blue dots represent those of the the original model, and the cyan dots represent their overlaps. In the first row, we label a classification-relevant area (e.g., the neck of the bird) using red circles. Within the red circle in the second-column image, the heatmap is more smooth, while in the third-column image, there are two obvious heat spots. In the fourth-column image, we notice that the original gradient would lead the adversarial attack to add more perturbations on this classification-relevant area, while the *Fake Gradient* would not. In the second row, we label the classification-irrelevant area (e.g., grassland) using red circles. We notice that the second-column image has less smooth heatmap than that in the third column. The fourth-column image shows that the fake gradient would result in perturbations added to the irrelevant area. In summary, by observing both of the two fourth-column images, the cyan dots are rare, which means that *Fake Gradient* can successfully mislead the adversarial attack to add perturbations to the areas irrelevant to the classification task.

We further evaluate the overall defense performance of *Fake Gradient* in terms of the defense success rate, as compared to the random weight approach. Instead of evaluating the classification accuracy based on the original perturbations, we use *Fake Gradient* to modify the 7 DNN models and employ *DeepFool* [19] to directly attack the modified models. The results are shown in Table 1. We observe that the defense success rate is very high ($\geq$79.8%) by using *Fake Gradient* while the random weight approach achieves very low success rate (i.e., $\leq$ 22.5%), because it cannot cause the adversarial attack algorithm to generate insensitive perturbations.

We also evaluate the L2 Norm of the adversarial samples to examine the detail of how *Fake Gradient* defends against the adversarial attack. We collect all the L2 Norm values for each test image and calculate the average, which are shown in Table 1. We observe that *Fake Gradient* causes the adversarial attack to generate fewer perturbations than the original and the random weight approaches.

5.4 Overhead

One of the advantages of *Fake Gradient* is the low protection overhead. First, comparing with the data encryption [4, 6, 33], distillation [21, 22], and adversarial training [8, 14, 26] approaches, *Fake Gradient* reduces huge protection overhead by eliminating the need of re-training. Second, comparing with the data encryption and data filtering methods, *Fake Gradient* only introduces little processing overhead in the model inference, since it only doubles the weights

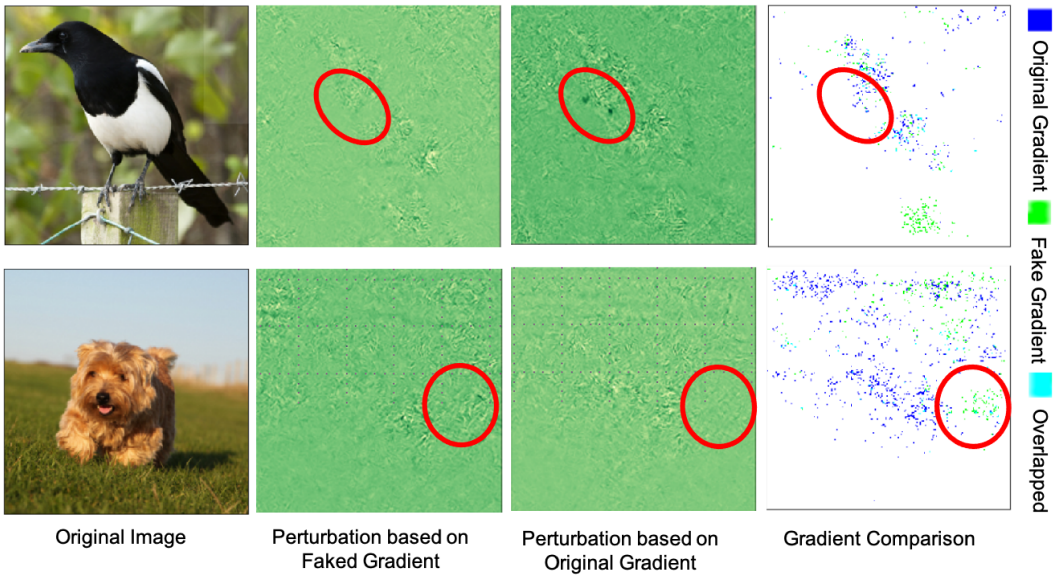

Figure 6: The intermediate results of the experiment with ResNet34.

of the last fully connected layer. Considering the huge portion of the convolution layers in the DNN models, the portion of the model size increased by *Fake Gradient* is small. Figure 7 compares the model size of the original model and the new model generated by *Fake Gradient*. We observe that size of the new model is almost the same as the original model, with only small increase. Furthermore, Figure 8 shows the average processing time for the 7 DNN models with and without *Fake Gradient*. We observe that *Fake Gradient* only increases less than 5% of the original inference time.

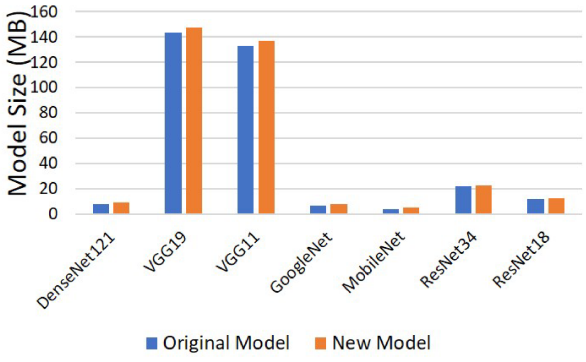

Figure 7: The comparison of model size.

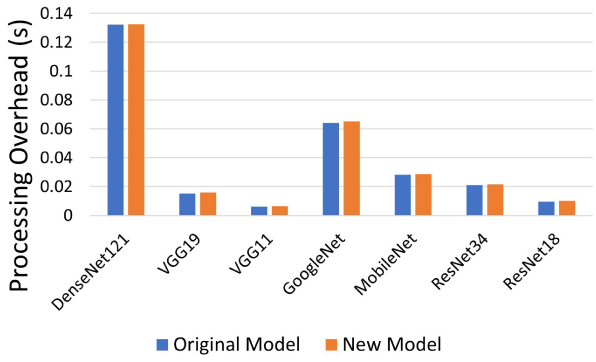

Figure 8: The comparison of processing overhead.

6 CONCLUSION

Targeting the privacy and security issues in the DNN-based image classification applications, we have developed the *Fake Gradient* framework, which can mislead the attacker to fake outputs and further use the fake gradient to prevent adversarial attacks. In particular, *Fake Gradient* adds new nodes and weights in the output layer (i.e., the last fully connected layer) of the DNN model, which cause the adversarial attack algorithm to generate significantly less effective perturbations. We evaluate *Fake Gradient* using the ImageNet dataset on 7 popular DNN models. The results show that *Fake Gradient* is effective in protecting privacy and defending against adversarial attacks in image classification applications. The project repository of *Fake Gradient* is located at https://github.com/hwsel/FakeGradient.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their constructive feedback. This work was partially supported by the National Science Foundation under award 1912593 and the Air Force Research Lab (AFRL) under Grant No. FA87501820058.

REFERENCES

[1] 2017. Cyber Security Firm Uses a 3D Printed Mask to Fool iPhone X's Facial Recognition Software. https://3dprint.com/194079/3d-printed-mask-iphone-x-face-id/

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283.

[3] Paolo Arena, Adriano Basile, Maide Bucolo, and Luigi Fortuna. 2003. Image processing for medical diagnosis using CNN. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 497, 1 (2003), 174–178.

[4] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning (ICML)*. 812–821.

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (S & P)*. 39–57.

[6] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *ACM Conference on Computer and Communications Security (CCS)*. 395–412.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 248–255.

[8] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research* 17, 1 (2016), 2096–2030.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4700–4708.

[13] Roberts Jeff. 2019. Look how easy it is to fool facial recognition – even at the airport. https://fortune.com/2019/12/12/airport-bank-facial-recognition-systems-fooled/.

[14] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *IEEE Symposium on Security and Privacy (S & P)*. 656–672.

[15] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2015. Visualizing and understanding neural models in NLP. *arXiv preprint arXiv:1506.01066* (2015).

[16] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *Network and Distributed Systems Security Symposium (NDSS)*.

[17] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. 2015. A unified gradient regularization family for adversarial examples. In *IEEE International Conference on Data Mining (ICDM)*. 301–309.

[18] Sébastien Marcel and Yann Rodriguez. 2010. Torchvision the machine-vision package of torch. In *ACM international conference on Multimedia (MM)*. 1485–1488.

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: a simple and accurate method to fool deep neural networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*. 2574–2582.

[20] Sharnil Pandya, Hemant Ghayvat, Ketan Kotecha, Mohammed Awais, Saeed Akbarzadeh, Prosanta Gope, Subhas Chandra Mukhopadhyay, and Wei Chen. 2018. Smart home anti-theft system: A novel approach for near real-time monitoring and smart home security for wellness protocol. *Applied System Innovation* 1, 4 (2018), 42.

[21] Nicolas Papernot and Patrick McDaniel. 2017. Extending defensive distillation. *arXiv preprint arXiv:1705.05264* (2017).

[22] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (S & P)*. 582–597.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[24] Ravi Prakash and Premkumar Chithaluru. 2021. Active security by implementing intrusion detection and facial recognition. *Nanoelectronics, Circuits and Communication Systems* (2021), 1–7.

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4510–4520.

[26] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free! *arXiv preprint arXiv:1904.12843* (2019).

[27] Uri Shaham, James Garritano, Yutaro Yamada, Ethan Weinberger, Alex Cloninger, Xiuyuan Cheng, Kelly Stanton, and Yuval Kluger. 2018. Defending against adversarial images using basis functions transformations. *arXiv preprint arXiv:1803.10840* (2018).

[28] Uri Shaham, Yutaro Yamada, and Sahand Negahban. 2015. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).

[29] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2019. A general framework for adversarial examples with objectives. *ACM Transactions on Privacy and Security (TOPS)* 22, 3 (2019), 1–30.

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

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[32] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[33] Liyao Xiang, Haotian Ma, Hao Zhang, Yifan Zhang, Jie Ren, and Quanshi Zhang. 2019. Interpretable complex-valued neural networks for privacy protection. *arXiv preprint arXiv:1901.09546* (2019).

[34] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017).

[35] Abdulsalam Yassine, Shailendra Singh, M Shamim Hossain, and Ghulam Muhammad. 2019. IoT big data analytics for smart homes with fog and cloud computing. *Future Generation Computer Systems* 91 (2019), 563–573.

[36] Zhe Zhou, Di Tang, Xiaofeng Wang, Weili Han, Xiangyu Liu, and Kehuan Zhang. 2018. Invisible mask: Practical attacks on face recognition with infrared. *arXiv preprint arXiv:1803.04683* (2018).