fakeWeather: Adversarial Attacks for Deep Neural Networks Emulating Weather Conditions on the Camera Lens of Autonomous Systems

Alberto Marchisio1,*, Giovanni Caramia2,*, Maurizio Martina2, Muhammad Shafique3
1 Technische Universität Wien, Vienna, Austria 2 Politecnico di Torino, Turin, Italy 3 New York University, Abu Dhabi, UAE
Email: alberto.marchisio@tuwien.ac.at, giovanni.caramia@studenti.polito.it maurizio.martina@polito.it, muhammad.shafique@nyu.edu

Abstract—Recently, Deep Neural Networks (DNNs) have achieved remarkable performances in many applications, while several studies have enhanced their vulnerabilities to malicious attacks. In this paper, we emulate the effects of natural weather conditions to introduce plausible perturbations that mislead the DNNs. By observing the effects of such atmospheric perturbations on the camera lenses, we model the patterns to create different masks that fake the effects of rain, snow, and hail. Even though the perturbations introduced by our attacks are visible, their presence remains unnoticed due to their association with natural events, which can be especially catastrophic for fully-autonomous unmanned vehicles. We test our proposed fakeWeather attacks on multiple Convolutional Neural Network and Capsule Network models, and report noticeable accuracy drops in the DNN predictions. While several works of adversarial phenomena. Hence, an image captured in such conditions represents a plausible image that can be processed by the DNN-based algorithm.

A. Target Research Problem and Associated Challenges

The key objective for an adversarial attack and its applicability in practical use-cases consists of not being recognized as adversarial, but rather as common/plausible. The most intuitive approach is to inject a very limited amount of perturbations, with the goal of making the differences between the clean images and the adversarial images imperceptible to the human eye. This approach has been adopted by several works, including Luo et al. [14], Croce et al. [15], and Marchisio et al. [16]. However, the attacker needs to have access to a set of information, including DNN model architecture and parameters, inputs, and outputs (i.e., in white-box settings), or only inputs and outputs (in black-box settings). Even the most advanced decision-based black-box attacks such as HopSkipJumpAttack [17] and FaDeC [18] still have access to the DNN predicted output class for each image. However, in practical cases, it may be very complicated to obtain such information, due to the protection mechanisms applied by the DNN-based system developers [19]. Moreover, another key limitation resides in the fact that even the most query-efficient algorithms [20] need to perform a certain number of queries (i.e., inference passes) to generate the adversarial perturbation, which may not be practical in case of stringent real-time constraints, because of the latency overhead caused by the queries.

Due to these limitations of the adversarial attacks that aim at introducing imperceptible perturbations compared to the original images, our approach follows a different strategy (see Figure 1). Our novel idea is to introduce perturbations to the input image in such a way that it is not considered as adversarial, since it resembles a natural condition captured by the camera. While the differences between the clean image and the adversarial image can be noticed, the adversarial image itself is hardly categorized as “adversarial”, since it simply captures a plausible natural condition. The reason is based on the fact that traditional adversarial machine learning takes into account the comparison between the adversarial image and the original image. However, in real-world practice,
it is impossible to obtain such a comparison, since the only accessible image is the one recorded by the camera. Noticeably, our approach is advantageous compared to previous works, since it is conducted in what we call a true black-box setting, i.e., in a scenario in which the attacker has no information about the DNN model architecture and parameters, nor its outputs. The only information required is the size of the input image, for generating an adversarial mask of that size. Moreover, our attack does not require any query, thus it can easily be applied at run time.

\begin{figure}[h!]
\centering
\includegraphics[width=0.8\textwidth]{fig1.png}
\caption{\textit{fakeWeather} attacks functionality.}
\end{figure}

\subsection{B. Our Novel Contributions}

Towards this, we observe how natural weather conditions, such as rain, snow, and hail, are perceived by the camera. We exploit this observation by designing \textit{fakeWeather} attack algorithms that emulate these effects on the camera lens. An overview of its functionality is shown in Figure 2. Our methodology can be used not only as an adversarial attack to mislead the DNN, but also as a data augmentation approach for reinforcing the DNN training under these conditions. Our contributions can be summarized as follows:

- We observe several images of natural weather events that affect the camera (Section III-B), and identify the patterns that are more commonly present in such images (Section III-C).
- By only knowing the image size, we design three \textit{fakeWeather} masks that fake the effect of such weather conditions on the camera lens (Sections III-D and III-F).
- We evaluate the \textit{fakeWeather} attacks on multiple DNN models (LeNet-5, ResNet-32, CapsNet) for the CIFAR-10 dataset, and obtain a success rate of the attacks varying between 30\% and 82.5\% (Section IV).

Before proceeding to the technical sections, we provide an overview of the adversarial attacks and the related works in Section II.

\section{II. BACKGROUND AND RELATED WORKS}

The purpose of the most common adversarial attack algorithms, such as gradient-based attacks \cite{brown2017adversarial}, is to introduce some perturbations that induce a decision boundary cross in the DNNs, and therefore lead to a misclassification. Examples of such attacks include the Fast-Gradient Sign Method (FGSM) \cite{goodfellow2014explaining}, DeepFool \cite{moosavi2016deepfool}, the Projected Gradient Descent (PGD) \cite{kurakin2016adversarial}, and the Carlini & Wagner attack \cite{carlini2017towards}. Other classes of attacks in which the perturbations were inserted only in a small set of pixels or only in one pixel were proposed by Narodytska et al. \cite{narodytska2017black} and Su et al. \cite{su2018robust}, respectively. Concurrently, Moosavi-Dezfooli et al. \cite{moosavi2016universal} proposed image-agnostic universal perturbations that are applied to every sample, and Zhang et al. \cite{zhang2019theoretically} generated different adversarial perturbations for each target class.

In black-box settings, several works were conducted. Kurakin et al. \cite{kurakin2016adversarial} proposed a method that crafts adversarial examples in the physical world by taking the images from a cell-phone camera. Moreover, taking into account the high-saliency and low-distortion path, Gragnaniello et al. \cite{gragnaniello2018uniattack} introduced an attack that improves the perceptual quality of the adversarial image.

Several attacks have been designed for real-world settings that incorporate so-called environmental perturbations. Brown et al. \cite{brown2017adversarial} generated adversarial patches, i.e., image-independent patches, to be placed anywhere inside the original image to mislead the DNNs. Following a similar approach, Eykholt et al. \cite{eykholt2018robust} added stickers to road signs to fool the traffic sign recognition system, while Sharif et al. \cite{sharif2017accessorize} added glasses to faces to fool the face recognition system. Man et al. \cite{man2019tricking} proposed GhostImage attacks, in which the adversarial patterns are inserted into the camera systems through a projector.

Focusing on more closely related approaches to our work, other methods in which DNN models are fooled due to atmospheric phenomena were proposed. Temel et al. \cite{temel2019characterizing} analyzed several challenging conditions, including snow and rain, for traffic sign detection systems, and collected them into their proposed CURE-TSD-Real dataset. Zhai et al. \cite{zhai2019adversarial} simulated various rainy situations using a gradient-based rain generation process. However, there are clear differences compared to our \textit{fakeWeather} attacks. Both these two related works inject perturbations in the long-range, i.e., relatively...
far from the camera, while our methodology introduces perturbations in the close proximity of the camera lens. Unlike other methods in the related works, our approach is non-invasive, since it does not require any modification in the real world, but it only modifies some pixel intensities of the images without interfering with the underlying DNN processing.

III. fakeWeather Attack Design

A. Problem Formulation and Assumptions

Taking into account the previous discussions, we propose the fakeWeather methodology. An overview of its functionality is shown in Figure 3. The final goal is to generate a finite set of perturbations with certain patterns which resemble the effect of natural weather events. Hence, such patterns are crafted by faking that the camera lens is dirty due to atmospheric conditions (such as rain, snow, and hail). After observing their effects on several examples in the real world, the common patterns are extracted and reproduced to generate the perturbation masks. The attack is conducted in what we call a true black-box setting, i.e., assuming that:

- the adversary has no information about the DNN model architecture, its parameters, and its output;
- the only information available for the attacker is the size of the input images.

B. Observation of Weather Conditions

The fakeWeather attacks are performed through the introduction of drops of water and snowflakes. A typical water drop has a spherical shape, while a snowflake has a hexagonal shape. However, in practical use-cases, these weather conditions do not represent the main focus of the camera. A camera captures the effects of rain and snow in a different way, which results into a set of blurry dots that are overlapped to the image. For instance, if we consider the use case of vision for smart mobility, the camera can be placed either outside of the vehicle (and hence exposed to the weather conditions), or inside the vehicle but in close proximity to the window. Without loss of generality, we can model a drop or a snowflake as a single pixel w.r.t. the image of $h \cdot l$ pixels, where $h$ and $l$ represent the height and length, respectively.

C. Pattern Extraction and Mask Generation

According to the previous considerations, the fakeWeather methodology extends the formulation of the One Pixel Attack [27], in which the perturbation of a single pixel is defined as a tuple of 5 elements $(x, y, r, g, b)$ where:

- $(x, y)$ represent the coordinates of the pixel to be modified;
- $(r, g, b)$ indicate the color of the pixel in RGB format.

Therefore, an adversarial pattern combines multiple pixel attacks, in which the perturbation introduced on the pixel $i$ can be written as in Equation (1). An example of the corresponding mask of an adversarial pattern is shown in Figure 4.

$$\text{pixel}_i = (x_i, y_i, r_i, g_i, b_i)$$

The colors, i.e., the values assumed by $(r_i, g_i, b_i)$, are determined according to the weather condition:

- rain: $(r_r, g_r, b_r) = (208, 209, 214)$
- snow and hail: $(r_s, g_s, b_s) = (249, 242, 242)$

For each type of fakeWeather attack (i.e., rain, snow, and hail), specific patterns are generated. Common patterns are extracted from real images and reproduced to form the set of pixel coordinates $(x_i, y_i)$ that belongs to the attack mask. Once generated, the same mask is applied to all the images under attack.

D. fakeRain Attack

The mask employed in the fakeRain attack is designed based on the combination of several water drops. In the real world, the camera lens can be soiled due to the rain, where the water droplets make up different patterns. It is possible to recognize three real-case scenarios, which can be categorized as agglomerate of drops, drop patch, and drop lines. As shown in Figure 5, the next step consists of modeling these patterns in terms of pixel coordinates that are perturbed.

The Agglomerate Pattern can be modeled by combining together 5 pixels to form a cross sign, according to the sketch in Figure 5a and Algorithm 1. The Patch Pattern (see Figure 5b) can have three different shapes, namely the vertical patch, which can be modeled as two consecutive pixels that
Fig. 5. Several patterns of water drops observed from the real environment. i) agglomerate of drops, ii) water drop patch, iii) drop lines.

Figure 6 illustrates the graphical representation of the three patterns: (a) Agglomerate Pattern, (b) Patch Patterns, and (c) Line Pattern.

Algorithm 1: Agglomerate Pattern

| input: Coordinate \((x_0, y_0)\) |
| output: Agglomerate Pattern \(P_a\) |

\[
\begin{align*}
P_a &= \emptyset \\
\text{for } i &\leftarrow 0 \text{ to } 2 \\
\text{for } j &\leftarrow 0 \text{ to } 2 \\
\text{if } (i + j = 0 \lor i + j = 2 \lor i + j = 4) &\text{ then} \\
P_a &\leftarrow \text{pixel}_{k} = (x_0 + i, y_0 + j, r, g, b) \\
k &\leftarrow k + 1 \\
\text{end} \\
\text{end} \\
\end{align*}
\]

Moreover, in rainy conditions, we can notice that the water drops tend to concentrate in the bottom corners of the image. Hence, to emulate this effect, in the fakeRain attack, a V-shape is created to divide the image into two regions (see the example in Figure 7). Below the V, several agglomerate patterns are densely concentrated. Above the V, path and line patterns are more sparsely distributed. Algorithm 4 describes the procedure for generating the fakeRain mask. Note that it is a three-step process in which (i) several agglomerate patterns are added (see line 2 of Algorithm 4), (ii) other agglomerate patterns are added if the coordinate is below the V (line 5 of Algorithm 4), and (iii) patch patterns of different types and line patterns are added above the V (line 7 of Algorithm 4).

E. fakeSnow Attack

The design of the fakeSnow attack is based on the assumption that a snowflake can be modeled as a single pixel, since the dimension of each snowflake is relatively small, as observed in Figure 8. According to these considerations, the snow pattern \(P_s\) consists of a single pixel, which can be modeled as in Equation 2 where \((x_0, y_0)\) represents the coordinate in which the snow pattern is constructed.
Algorithm 2: Patch Pattern

```plaintext
input : Coordinate \((x_0, y_0)\), Type \(t\)
output: Patch Pattern \(P_p\)
1 \(P_p = \emptyset\)
2 switch \(t\) do
3 case 0 do // Vertical Patch
4 for \(j \leftarrow 0\) to 1 do
5 \(P_p \leftarrow \text{pixel}_j = (x_0, y_0 + j, r_r, g_r, b_r)\)
6 end
7 end
8 case 1 do // Diagonal Patch
9 \(k = 0\)
10 for \(i \leftarrow 0\) to 1 do
11 for \(j \leftarrow 0\) to 1 do
12 if \((i + j = 1)\) then
13 \(P_p \leftarrow \text{pixel}_k = (x_0 + i, y_0 + j, r_r, g_r, b_r)\)
14 \(k \leftarrow k + 1\)
15 end
16 end
17 end
18 case 2 do // Two Dots Patch
19 for \(j \leftarrow 0\) to 1 do
20 \(P_p \leftarrow \text{pixel}_j = (x_0, y_0 + 2 \cdot j, r_r, g_r, b_r)\)
21 end
22 end
23 end
```

Algorithm 3: Line Pattern

```plaintext
input : Coordinate \((x_0, y_0)\), Length \(n\)
output: Line Pattern \(P_l\)
1 \(P_l = \emptyset\)
2 for \(j \leftarrow 0\) to \(n - 1\) do
3 \(P_l \leftarrow \text{pixel}_j = (x_0, y_0 + j, r_r, g_r, b_r)\)
4 end
```

Algorithm 4: fakeRain Mask Generation

```plaintext
input : Image size: length \(l\) and hight \(h\)
output: fakeRain Mask \(M_r\)
1 \(M_r = \emptyset\)
2 \(M_r \leftarrow P_a(\{0, ..., l - 3\}, 0)\)
// use many agglomerate patterns in the first line
3 for \((i, j) \in \{(0, ..., l - 3), \{0, ..., h - 3\}\}\) do
4 if \((i + j < \frac{l - 1}{4}) \lor (i - j < \frac{h - 1}{4})\) then
5 \(M_r \leftarrow \text{polygon}(i, j) \lor \{\}
// sparsely add agglomerate patterns below the V
6 else
7 \(M_r \leftarrow \text{polygon}(i, j, t) \lor \text{polygon}(i, j, n) \lor \{\}
// sparsely add patch patterns or line patterns above the V
8 end
```

Algorithm 5: fakeSnow Mask Generation

```plaintext
input : Image size: length \(l\) and hight \(h\)
output: fakeSnow Mask \(M_s\)
1 \(M_s = \emptyset\)
2 for \(j \in \{0, 2, 4, ..., h - 2\}\) do
3 if \((j < \frac{h - 1}{2}) \lor (j > \frac{h - 1}{2} - 1)\) then
4 // upper and lower parts
5 if \(j \equiv 0\) mod 4 then
6 \(M_s \leftarrow \text{polygon}(0, 3, 6, 9, ..., l - 2), j + 1\)
7 else // skip some snow patterns
8 \(M_s \leftarrow \text{polygon}(0, 6, 12, ..., l - 2), j + 1\)
9 else // middle part
10 \(M_s \leftarrow \text{polygon}(0, 3, 6, 9, ..., l - 2), j + 1\)
// add dense snow patterns
11 end
```

F. fakeHail Attack

Compared to the snow, a hail scenario produce relatively larger ice balls perceived by the camera, as shown in Figure 10. Hence, the hail pattern is not modeled as a single pixel, but as an agglomerated of 8 pixels, as described in Algorithm 6

Since the hail patterns appear irregularly, the fakeHail mask can be generated through a collection of hail patterns, as

Fig. 8. Several snowflakes observed, which can be modeled as single dots.

Another key feature noticed from the observation of real images is that the snowflakes are more densely concentrated in close proximity to the horizon line. In practice, this effect can be modeled by cutting the image into three parts through
Algorithm 7: fakeHail Mask Generation

\begin{verbatim}
input : Image size: length \( l \) and height \( h \)
output: fakeHail Mask \( M_h \)
1 \( M_h = \emptyset \)
2 \( \text{for} \ (i, j) \in \{(0, ..., l - 4), \{0, ..., h - 4\}\} \text{do} \)
3 \( M_h \leftarrow F_h(i, j) \lor \{\} \)
4 \( \text{end} \)
\end{verbatim}

The LeNet, which is composed of two convolutional layers and two fully-connected layers followed by a softmax layer, has been trained for 200 epochs, using a batch size of 128, weight decay 0.0001, and a learning rate scheduler that progressively reduces its value from 0.05 to 0.0004. The 32-layer ResNet has been trained for 200 epochs, using a batch size of 128, weight decay 0.0001, and a learning rate scheduled to decrease from 0.1 to 0.001. The CapsNet, composed of a convolutional layer, a primary capsule layer, and a dynamic routing layer, has been trained for 200 epochs with a batch size of 64 and a learning rate equal to 0.001. For clean test images, we measure the accuracy values of 74.88\%, 92.31\%, and 79.82\%, for the LeNet, ResNet, and CapsNet, respectively.

Afterwards, the \textit{fakeWeather} masks have been applied to 200 testing samples and the attack success rate has been evaluated for every attack type (i.e., \textit{fakeRain}, \textit{fakeSnow} and \textit{fakeHail}) and every DNN model. The training, as well as the implementation of the \textit{fakeWeather} attacks and their evaluation, has been carried out using the Keras framework [40] with the TensorFlow [41] back-end, and executed on an ML-workstation equipped with two Nvidia GeForce RTX 2080 Ti GPUs.

Algorithm 6: Hail Pattern

\begin{verbatim}
input : Coordinate \((x_0, y_0)\)
output: Hail Pattern \( P_h \)
1 \( P_h = \emptyset \)
2 \( k = 0 \)
3 \( \text{for} \ i \leftarrow 0 \ \text{to} \ 3 \ \text{do} \)
4 \( \quad \text{for} \ j \leftarrow 0 \ \text{to} \ 3 \ \text{do} \)
5 \( \quad \quad \text{if} \ (i = j \land i < 2) \lor (i + j = 3) \lor (i = 2 \land j \neq 2) \)
6 \( \quad \quad \quad \text{then} \ P_h \leftarrow \text{pixel}_k = (x_0 + i, y_0 + j, r_s, g_s, b_s) \)
7 \( \quad \quad \quad \quad k \leftarrow k + 1 \)
8 \( \quad \text{end} \)
9 \( \text{end} \)
10 \( \text{end} \)
\end{verbatim}

The \textit{fakeWeather} mask has been generated to observe the conditions that lead to the design of the hail pattern.

Algorithm 6: Hail Pattern

\begin{verbatim}
input : Coordinate \((x_0, y_0)\)
output: Hail Pattern \( P_h \)
1 \( P_h = \emptyset \)
2 \( k = 0 \)
3 \( \text{for} \ i \leftarrow 0 \ \text{to} \ 3 \ \text{do} \)
4 \( \quad \text{for} \ j \leftarrow 0 \ \text{to} \ 3 \ \text{do} \)
5 \( \quad \quad \text{if} \ (i = j \land i < 2) \lor (i + j = 3) \lor (i = 2 \land j \neq 2) \)
6 \( \quad \quad \quad \text{then} \ P_h \leftarrow \text{pixel}_k = (x_0 + i, y_0 + j, r_s, g_s, b_s) \)
7 \( \quad \quad \quad \quad k \leftarrow k + 1 \)
8 \( \quad \text{end} \)
9 \( \text{end} \)
10 \( \text{end} \)
\end{verbatim}

The LeNet, which is composed of two convolutional layers and two fully-connected layers followed by a softmax layer, has been trained for 200 epochs, using a batch size of 128, weight decay 0.0001, and a learning rate scheduler that progressively reduces its value from 0.05 to 0.0004. The 32-layer ResNet has been trained for 200 epochs, using a batch size of 128, weight decay 0.0001, and a learning rate scheduled to decrease from 0.1 to 0.001. The CapsNet, composed of a convolutional layer, a primary capsule layer, and a dynamic routing layer, has been trained for 200 epochs with a batch size of 64 and a learning rate equal to 0.001. For clean test images, we measure the accuracy values of 74.88\%, 92.31\%, and 79.82\%, for the LeNet, ResNet, and CapsNet, respectively.

Afterwards, the \textit{fakeWeather} masks have been applied to 200 testing samples and the attack success rate has been evaluated for every attack type (i.e., \textit{fakeRain}, \textit{fakeSnow} and \textit{fakeHail}) and every DNN model. The training, as well as the implementation of the \textit{fakeWeather} attacks and their evaluation, has been carried out using the Keras framework [40] with the TensorFlow [41] back-end, and executed on an ML-workstation equipped with two Nvidia GeForce RTX 2080 Ti GPUs.

B. \textit{fakeWeather} Attacks Evaluation

Table I reports the results for the \textit{fakeRain}, \textit{fakeSnow} and \textit{fakeHail} attacks in terms of Adversarial Success Rate (ASR), which corresponds to the ratio between the misclassified examples and all the tested examples. The results are compared with the state-of-the-art 1-pixel, 3-pixel, and 5-pixel attacks proposed by Su et al. [27]. Moreover, Figure 12 shows a collection of adversarial examples generated with the \textit{fakeWeather} attacks.

\textbf{fakeRain Evaluation}

The \textit{fakeRain} attack is successful for the LeNet and the ResNet, since their ASRs are 72\% and 67\%, respectively. The
TABLE I
Evaluation of the Adversarial Success Rate (ASR) for the the LeNet, the ResNet, and the CapsNet on the CIFAR-10 dataset. Our proposed fakeWeather attacks have been compared to the 1-pixel, 3-pixel, and 5-pixel attacks [27].

| ASR on Attack | LeNet | ResNet | CapsNet |
|---------------|-------|--------|---------|
| 1-pixel [27]  | 63%   | 34%    | 19%     |
| 3-pixel [27]  | 92%   | 79%    | 39%     |
| 5-pixel [27]  | 93%   | 79%    | 36%     |
| fakeRain (ours) | 72%   | 67%    | 36%     |
| fakeSnow (ours) | 75.5% | 79.5%  | 30%     |
| fakeHail (ours) | 82.5% | 78.5%  | 63%     |

ResNet results slightly more robust than the LeNet, due to its deeper structure. The ASR falls to 36% for the CapsNet, since its architecture that groups the neurons into capsules, along with the dynamic routing, helps to better encode the spatial relations between features of the images. The example in Figure 12a shows the image of a deer on which the fakeRain mask is applied. All the three DNN models erroneously classify it as a “bird”, while its clean version is correctly classified as a “deer”.

fakeSnow Evaluation
For the fakeSnow attack, the relations between the ASRs of the three DNN models are similar to the observations made for the fakeRain attack, in which the CapsNet is more robust than the other CNNs. However, the ASR results are higher for the ResNet, compared to the LeNet. The example in Figure 12c showing a frog with the fakeSnow mask is correctly classified by the CapsNet, while it is incorrectly classified as a “cat” by the ResNet and as a “truck” by the LeNet. Its clean version is correctly classified as a “frog” by all the DNNs. The horse in Figure 12f is correctly classified by the ResNet and the CapsNet, while the LeNet classifies it as a “deer”.

fakeHail Evaluation
The ASR relative to the fakeHail attack is significantly higher than the previous attacks, in particular for the CapsNet. Due to the relatively large perturbations imposed by the hail patterns (i.e., 8-pixel perturbations), the fakeHail mask can break the spatial relations learned by the CapsNet and lead to many misclassified samples. The example in Figure 12c represents a ship with the fakeHail mask that is incorrectly classified as an “airplane” by the LeNet and CapsNet, and as a “truck” by the ResNet. The image in Figure 12c is incorrectly classified as a “cat” by the LeNet, as a “deer” by the ResNet, and as a “frog” by the CapsNet, despite showing an airplane.

C. Case Studies: Output Probability Variations under fakeWeather attacks.
Towards a more comprehensive evaluation, we analyze the output probability variations when different types of fakeWeather attacks are applied to the LeNet, ResNet, and CapsNet models. For reference, the 10 classes of the CIFAR-10 dataset are associated with a digit 0 – 9 according to the convention in Table II.

TABLE II
Class labels for the CIFAR-10 dataset [39].

| # Class | Class  |
|---------|--------|
| 0       | airplane |
| 1       | automobile |
| 2       | bird |
| 3       | cat |
| 4       | deer |
| 5       | dog |
| 6       | frog |
| 7       | horse |
| 8       | ship |
| 9       | truck |

Figure 13 shows how the image of a “truck” of the CIFAR-10 dataset is classified, for different fakeWeather attacks and different DNN models. The clean image is correctly classified as the class 9, i.e., “truck” by the LeNet, despite having a relatively low confidence (see pointer 1 in Figure 13b). When each of the fakeWeather masks is applied, the LeNet predicts the image as a “frog” with quite high confidence (see pointer 2 in Figure 13). The probability variations for the ResNet assume a different behavior. While the clean image is correctly classified with high confidence (see pointer 3 in Figure 13c).
in Figure 13), the fakeWeather attacks produce different outcomes. With the fakeRain mask the image is classified as an “automobile” by the ResNet (see pointer 4 in Figure 13c), with the fakeSnow mask the highest probability belongs to the class “bird” (see pointer 5 in Figure 13c), and the adversarial fakeHail image is classified as an “airplane” by the ResNet (see pointer 6 in Figure 13c). The output probabilities for the CapsNet, while they are more concentrated in the middle values, i.e., 1/10, show that the clean image is correctly classified (see pointer 7 in Figure 13d), while for all the fakeWeather attacks, the highest probability belongs to the class “horse” (see pointer 8 in Figure 13d).

D. Results Discussion and Comparison

To summarize, given the above-discussed results, we can make the following considerations:

- All the fakeWeather attacks produce a high ASR for the LeNet and ResNet (ASR > 65%).
- The fakeHail attack is the strongest, since it achieves an ASR equal to 63% for the CapsNet and higher for the other DNNs.

Compared to the methods of Su et al. [27], our fakeWeather methods have higher ASR than the 1-pixel attack for every DNN model (see Table I). However, the 3-pixel and 5-pixel attacks have higher ASR than our methods. Note that the approach used by Su et al. is based on an evolutionary algorithm that requires several queries, while our methodology does not require any query. Yet, the ASR relative to the CapsNet for the fakeHail attack is 27% higher than the 5-pixel attack.

E. Future Outlooks and Applicability

From another perspective, our contributions, other than a methodology for generating adversarial attacks in real-time without queries, can be viewed as a data augmentation methodology for generating synthetic samples of weather conditions. We envision the possibility of enlarging the dataset with images that contain fakeWeather masks and train DNN-based classifiers more robustly to such atmospheric phenomena, in a similar way as the adversarial training’s functionality [24]. Since the only information required is the image size, its high scalability makes our fakeWeather attack methodology suitable to any vision-based outdoor application.
V. Conclusion

In this paper, we presented fakeWeather attacks, adversarial attacks for DNNs that emulate the natural weather conditions. Our methodology consists of observing a series of images that capture the effects of such conditions perceived by the camera lens, and modeling a set of patterns to create dedicated fakeRain, fakeSnow, and fakeHail masks as a collection of these patterns. Hence, these sets of perturbations make the adversarial image a plausible input to the DNN. Our proposed attack is conducted in true black-box settings, in which the adversary has no access to the DNN model, its parameters, and its output. The evaluation of fakeWeather attacks on different DNN models (Convolutional Neural Networks and Capsule Networks) highlights noticeable adversarial success rates.

ACKNOWLEDGMENT

This work has been supported in part by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien’s Faculty of Informatics and the UAS Technikum Wien. This work was also supported in parts by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG010, Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104, and Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

REFERENCES

[1] L. Jiao et al., “A survey of deep learning-based object detection,” IEEE Access, 2019.
[2] S. Kuutti et al., “A survey of deep learning applications to autonomous vehicle control,” arXiv, 2019.
[3] R. Lee et al., “Deep neural network–based enhancement for image and video streaming systems: A survey and future directions,” ACM Comput. Surv., 2021.
[4] K. He et al., “Deep residual learning for image recognition,” in CVPR, 2016.
[5] M. Capra et al., “Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead,” IEEE Access, 2020.
[6] M. Capra et al., “An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks,” Future Internet, 2020.
[7] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” in CCS, 2018.
[8] N. Dalvi et al., “Adversarial classification,” in KDD, 2004.
[9] D. Lowd and C. Meek, “Adversarial learning,” in KDD, 2005.
[10] C. Szegedy et al., “Intriguing properties of neural networks,” in ICLR, 2014.
[11] M. Shafique et al., “Towards energy-efficient and secure edge AI: A cross-layer framework ICCAD special session paper,” in ICCAD, 2021.
[12] S. Dave et al., “Special session: Towards an agile design methodology for efficient, reliable, and secure ML systems,” in VTS, 2022.
[13] M. Shafique et al., “Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead,” IEEE Design & Test, 2020.
[14] B. Luo et al., “Towards imperceptible and robust adversarial example attacks against neural networks,” in AAAI, 2018.
[15] F. Croce and M. Hein, “Sparse and imperceivable adversarial attacks,” in ICCV, 2019.
[16] A. Marchisio et al., “Capsattacks: Robust and imperceptible adversarial attacks on capsule networks,” arXiv, 2019.
[17] J. Chen et al., “Hopskipjumpattack: A query-efficient decision-based attack,” in SP, 2020.
[18] F. Khalid et al., “Fadec: A fast decision-based attack for adversarial machine learning,” in IJCNN, 2020.
[19] M. Xue et al., DNN Intellectual Property Protection: Taxonomy, Attacks and Evaluations (Invited Paper). 2021.
[20] D. Willmott et al., “You only query once: Effective black box adversarial attacks with minimal repeated queries,” arXiv, 2021.
[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in ICLR, 2015.
[22] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” arXiv, 2016.
[23] S.-M. Moosavi-Dezfooli et al., “Deepfool: A simple and accurate method to fool deep neural networks,” in CVPR, 2016.
[24] A. Madry et al., “Towards deep learning models resistant to adversarial attacks,” in ICLR, 2018.
[25] N. Carlini and D. A. Wagner, “Towards evaluating the robustness of neural networks,” arXiv, 2016.
[26] N. Narodytska and S. Kasisivvanathan, “Simple black-box adversarial attacks on deep neural networks,” in CVPRW, 2017.
[27] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” arXiv, 2017.
[28] S.-M. Moosavi-Dezfooli et al., “Universal adversarial perturbations,” in CVPR, 2017.
[29] C. Zhang et al., “CD-UAP: class discriminative universal adversarial perturbation,” in AAAI, 2020.
[30] D. Gragnaniello et al., “Perceptual quality-preserving black-box attack against deep learning image classifiers,” arXiv, 2019.
[31] T. B. Brown et al., “Adversarial patch,” arXiv, 2017.
[32] K. Eykholt et al., “Robust physical-world attacks on deep learning visual classification,” in CVPR, 2018.
[33] M. Sharif et al., “Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition,” in CCS, 2016.
[34] Y. Man, M. Li, and R. Gerdes, “Ghostimage: Remote detection under challenging conditions: A deeper look into performance variations and spectral characteristics,” in IEEE TITS, 2019.
[35] L. Zhai et al., “Towards deep learning models resistant to adversarial attacks,” in IJCNN, 2015.
[36] D. Temel, M.-H. Chen, and G. AlRegib, “Traffic sign detection under challenging conditions: A deeper look into performance variations and spectral characteristics,” in IEEE TITS, 2019.