Generation of Black-box Audio Adversarial Examples Based on Gradient Approximation and Autoencoders

PO-HAO HUANG, Department of Computer Science, National Tsing Hua University, Taiwan
HONGGANG YU and MAX PANOFF, Department of Electrical and Computer Engineering, University of Florida, USA
TING-CHI WANG, Department of Computer Science, National Tsing Hua University, Taiwan

Deep Neural Network (DNN) is gaining popularity thanks to its ability to attain high accuracy and performance in various security-crucial scenarios. However, recent research shows that DNN-based Automatic Speech Recognition (ASR) systems are vulnerable to adversarial attacks. Specifically, these attacks mainly focus on formulating a process of adversarial example generation as iterative, optimization-based attacks. Although these attacks make significant progress, they still take large generation time to produce adversarial examples, which makes them difficult to be launched in real-world scenarios. In this article, we propose a real-time attack framework that utilizes the neural network trained by the gradient approximation method to generate adversarial examples on Keyword Spotting (KWS) systems. The experimental results show that these generated adversarial examples can easily fool a black-box KWS system to output incorrect results with only one inference. In comparison to previous works, our attack can achieve a higher success rate with less than 0.004 s. We also extend our work by presenting a novel ensemble audio adversarial attack and testing the attack on KWS systems equipped with existing defense mechanisms. The efficacy of the proposed attack is well supported by promising experimental results.

CCS Concepts: • **Security and privacy → Software security engineering;**

Additional Key Words and Phrases: Adversarial examples, deep neural network, automatic speech recognition

ACM Reference format:
Po-Hao Huang, Honggang Yu, Max Panoff, and Ting-Chi Wang. 2022. Generation of Black-box Audio Adversarial Examples Based on Gradient Approximation and Autoencoders. *J. Emerg. Technol. Comput. Syst.* 18, 3, Article 59 (August 2022), 19 pages.
https://doi.org/10.1145/3491220

1 INTRODUCTION

Deep Neural Network (DNN) has achieved great progress in various security-crucial scenarios, including object detection, audio recognition and natural language processing[19, 20, 31]. Despite its popularity, recent works have shown that the DNN model is vulnerable to adversarial examples. These examples can force the target DNN model to output incorrect results by adding some

Authors' addresses: P.-H. Huang and T.-C. Wang, Department of Computer Science, National Tsing Hua University, No. 101, Section 2, Kuang-Fu Road, Hsinchu 30013, Taiwan; emails: kevin841006kg@gmail.com, tcwang@cs.nthu.edu.tw; H. Yu and M. Panoff, Department of Electrical and Computer Engineering, University of Florida, 327 Benton Hall, Gainesville, Florida, USA, 32611; emails: {honggang.yu, m.panoff}@ufl.edu.

1550-4832/2022/08-ART59 $15.00
https://doi.org/10.1145/3491220

elaborated perturbation to original inputs. Existing studies on adversarial examples mainly focus on image recognition tasks [4, 5, 18, 22, 24]. Nonetheless, adversarial examples in the audio domain have not received enough attention. The goal of these adversarial attacks is to add inaudible noises to the audio, which can mislead **Automatic Speech Recognition (ASR)** systems with high success rate. The adversarial attack can be divided into two categories: white-box attack and black-box attack. In the white-box attack [7], the attacker can have full knowledge of the target model, such as model structure or parameters. If the model is differentiable, then we can perform a gradient-based method to find the perturbation using back-propagation. In the black-box attack, the attacker has no access to the internal details of the target model but can observe the classification outputs (e.g., label, confidence score) while offering random inputs.

There are also some existing works that claim to propose defense methods to enhance the robustness of the target DNN model. The goal of these defense is to avoid the target model being misled by the adversarial examples. Similar to Reference [3], we roughly divide these defense methods into two categories: proactive defense and reactive defense. For proactive defense, the defender increases the model robustness by modifying the model parameters or structures during the training phase. The second category is reactive defense, where the defender adds the defense mechanism after the model is trained, and the weight parameters of the trained model cannot be modified.

Although recent adversarial attacks have made significant progress, they remain impractical in real-world ASR systems due to the following limitations:

- Due to economic and privacy concerns, an adversary usually cannot access the model details such as structure or parameters . Also, since the pre-processing procedure (e.g., **Mel-Frequency Cepstral Coefficients (MFCC)** calculation) of an KWS system is non-differentiable [40], it is difficult for an adversary to perform powerful white-box adversarial attacks.
- Though previous adversarial attacks can achieve high success rates and low perturbation on ASR systems, they often take too much time to craft adversarial examples, which makes them hard to perform a black-box adversarial attack in real-time scenario.
- The adversarial defense on ASR systems has already existed for a while, especially in the audio pre-processing domain. Previous adversarial attacks mainly focus on attacking the model without any defense mechanisms. When testing on the real-world system equipped with defense methods, these attacks still cannot bypass these defense successfully.

To address these challenges, in this article, we propose a novel adversarial attack framework that trains a neural network model (i.e., a generator model) to generate an adversarial example against a KWS system in the black-box setting. By using the generator model, we only need one inference to generate an adversarial example.

The main contributions of this article are summarized as follows:

- Unlike previous black-box attacks, which need a large amount of iterations to mislead the DNN model, we generate audio adversarial examples by using a generator model. After training the generator model, we only need one inference to perform a real-time attack against the black-box KWS systems.
- Since the attack is performed in the black-box setting, we cannot get the real gradient via back-propagation. To solve this problem, we use a gradient estimator to approximate the gradient of the loss with respect to the last second layer in the generator model. By doing so, we can still successfully train the generator model.
- We further extend our generator model in the proposed attack framework. We first present a novel ensemble attack that reformulates the objective function of the generator model by

combining the losses from multiple target models. The proposed attack can fool multiple KWS systems by a single adversarial example with high success rate. We then enhance the robustness of our generator model by reformulating the objective function of the generator model with additional loss terms against existing defense mechanisms.
- The experimental results show that our methods can reach faster generation speed and higher attack success rates when compared to previous black-box attack methods against KWS systems. Also, for ensemble attacks and attack enhancement, we can still achieve the satisfactory success rate.

We organize the rest of this article as follows. Section 2 reviews some existing KWS systems and some prior adversarial attack and defense works in the audio domain. Section 3 explains our attack methods against KWS systems. We show the experimental results in Section 4 and conclude this article in Section 5.

2 RELATED WORKS

2.1 Keyword Spotting System

A KWS system is often used for speech-based user interactions with some intelligent applications, such as "Alexa" or "Siri" in a smartphone. Unlike other ASR systems, the KWS system focuses on recognizing pre-defined keywords with a high recognition rate. Due to its low latency and small power consumption, a KWS system has been widely used in some embedded systems. This high applicability also encourages researchers to study KWS systems. It had first been mentioned in Reference [30] as early as 50 years ago.

A KWS system mainly contains two parts: feature extraction and audio classification. The input audio will first go through the feature extraction, in which MFCC is the commonly used techniques. By converting the original signal into spectral coefficients, we can extract the important features in the frequency domain. In the classification, we will perform inference by feeding the features into the classification model to make the final decision. Traditionally, Wilpon et al. [36] and Ros et al. [27] use **hidden Markov models (HMMs)** for KWS systems. Although this method gives a high recognition rate, the high computation overhead during inference is the major shortcoming. With the rise of deep learning, more and more research works like [12, 23, 34], use DNN models as the main classifier. For instance, Zhang et al. [40] provide **convolutional neural network (CNN)**, **deep neural networks (DNN)**, **recurrent neural network (RNN)**, **convolutional recurrent neural network (CRNN)**, and **depthwise separable convolutional neural network (DS-CNN)** as the classification model in a KWS system. In our work, we will use these models from Reference [40] as our target models.

2.2 Audio Adversarial Attacks

The audio adversarial attacks can be divided into two categories: white-box attack and black-box attack. In the white-box setting, Carlini et al. [7] reformulate the objective function by adding **connectionist temporal classification (CTC)** loss to original loss from Reference [6]. They fool the DeepSpeech [17], which is a speech to text model from Baidu in the target attack. However, the amount of perturbation is too large. To address this issue, Qin et al. [25] additionally uses the psychoacoustic principle of auditory masking to filter out the sensitive range of frequency from audio adversarial examples. Some works also mislead the model by attacking the white-box substitute model first and then transferring the adversarial examples to the black-box target model, such as Reference [28], which performs a non-target attack to the audio classifier and transfers to the different unseen models. However, the major problem is that we cannot ensure the attacking quality during the generation of adversarial examples, which increases the risk of failure. So

recently, most existing black-box attacks focus on attacking the target model directly. Alzantot et al. [2] attack a black-box KWS system by using a genetic algorithm, which adds a small random perturbation iteratively, and chooses the best result from the population. To perform the attack on a speech to text system, Taori et al. [29] enhance the previous method from Reference [2] by using gradient approximation.

The methods mentioned above take a long generation time for a single adversarial example by iteratively optimizing the problem to find the solution, which gives no possibility for attacking in real-time systems. One way to solve this problem is to perform a universal attack, which can manipulate all input data to the desired label by adding the same perturbation, such as Reference [35], which creates the universal adversarial perturbation against the audio classifier in the white-box setting. However, due to the high complexity to attack all input audio with one single perturbation, the attack success rate is lower than 50% in their experimental results. To retain high attack success rates and fast generation time, in this work, we train a generator model, which can map all input audio to their corresponding perturbations within the time of one inference. The generation process is similar to the works from References [8, 14]. However, Chang et al. [8] can only be performed in white-box setting, and Gong et al. [14] perform their attack for non-target cases. In this work, we can perform a target attack against a KWS system in real-time. Even more, we can perform our method in the black-box setting, which is more powerful than white-box attacks.

2.3 Audio Adversarial Defenses

The most common type of audio adversarial defense is to use input transformation [3]. The key idea of this method is to perform some data pre-processing methods on the raw data to destroy the adversarial perturbation before passing to the recognition model. Yang et al. [39] first performs local smoothing, quantization, and down-sampling to defend the adversarial example generated from Reference [2]. With higher quantization value, which rounds the amplitude of raw audio to the closest integer, the adversarial example with smaller perturbation will be disrupted but it also decreases the model accuracy for clean audio. To solve this problem, Rajaratnam et al. [26] ensemble different audio pre-processing methods together for detecting adversarial examples. They use the final prediction scores, which performs with and without the audio pre-processing method to train the binary detector. If the given input has a high similarity between the prediction scores with and without audio pre-processing, then it can be considered as clean audio.

3 METHODOLOGY

3.1 Overall Flow

In this part, we give an overview of our work. Our work is to train a generator model to generate the adversarial example against a KWS system in the black-box setting. When performing inference, the generator model will generate perturbations for each input audio. The generator model will first encode the audio into a low-dimensional latent vector and then decode it back to a perturbation that has the same dimension as the input. The inference process is defined as

$$\begin{aligned} g_\theta(x) : x \in X \rightarrow \delta, \\ x' = x + \delta, \end{aligned} \tag{1}$$

where θ denotes the weights of the generator model g, δ denotes the perturbation generated by the generator model, x' denotes the adversarial example, and X is the audio dataset. After finishing the training, we just need one inference to generate an adversarial example.

The training flow is similar to that of the traditional **Generative Adversarial Network (GAN)** [15]. The only difference is that we treat the KWS model as a discriminator, in which the weights

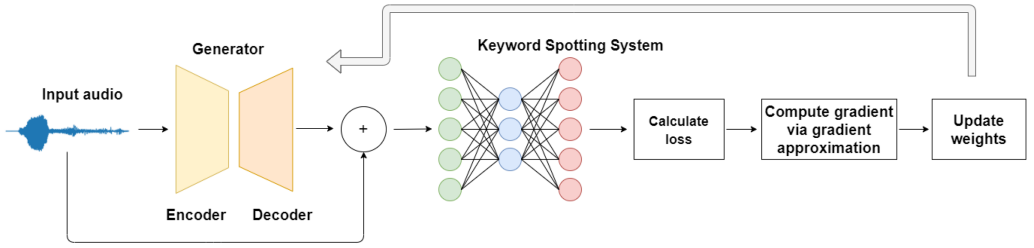


Fig. 1. Overall flow of the training procedure for the generator model.

are fixed. The training flow is shown in Figure 1. For each training iteration, we first perform inference on a batch of training data to the generator model to get the corresponding adversarial examples. Second, we feed these adversarial examples to our black-box KWS system to get the confidence scores and calculate the loss, which will be described in Section 3.3. Third, we update the weights of the generator model. However, due to black-box settings for the KWS system, we cannot update the weights by back-propagation. To solve this problem, we use gradient approximation, which will be introduced in Section 3.4, to simulate the gradient of the loss with respect to the weights. The training will stop after reaching the predefined max training epoch.

3.2 Generator Model Structures

We adopt two different model structures in this work. These models are an auto-encoder-based model. Unlike applications in the image domain, the audio domain has some different pre-processing ways to extract the features. **Fast Fourier transform (FFT)** and **short-time Fourier transform (STFT)** are the two most famous audio pre-processing strategies. So, we respectively use them to be the pre-processing layer in our two generator models. The details of the models are shown in Table 1, where *abs* denotes the absolute value function. We use *tanh* as the activation function.

3.2.1 FFT-NN. The first layer of the FFT-NN model is a FFT layer, which is used as a feature extraction stage. FFT can transform the sequence of values into different frequency components. After performing FFT, the original time-domain audio with length 16,000 will be converted to frequency-domain features with size 8,001. Then, we connect the FFT layer to three dense layers, where the sizes of the dense layers are 8,001, 200, 16,000, accordingly. The FFT and the first and second dense layers can be treated as an encoder, which learns the low-dimensional representation for the given data. We empirically observed that two dense layers are adequate for the encoder. The last layer can be treated as a decoder, which decodes the latent space features back to time-domain perturbation.

3.2.2 STFT-CNN. Different from the FFT-NN model, we use a STFT as a feature extraction stage. It first uses a sliding window to cut the audio into time frames and applies Fourier transform to extract the frequency and phase features for each frame. Compared to FFT, STFT keeps time-dimensional information, which makes it more powerful to extract useful features in audio. We set the frame step, frame length, and FFT size to 200, 800, 256, respectively.

After performing STFT, the original time-domain audio with length 16,000 will be converted to a 2D time-frequency spectrum feature with size 77×129 and fed into a CNN. The CNN is composed of three convolution layers, one max-pooling layer, which can be considered as an encoder, and one dense layer for the decoder. The filter width, filter height, input channels and output channels for layer one to three are (3,3,1,24), (3,3,24,24), (1,1,24,1), respectively. Pool height, pool width, and

Table 1. Two Kinds of Generator Model Structures

Structures	
FFT-NN	STFT-CNN
abs(FFT)	abs(STFT)
↓	↓
Dense(8001)	Conv(3,3,1,24)
↓	↓
Dense(200)	Conv(3,3,24,24)
↓	↓
Dense(16000)	Conv(1,1,24,1)
↓	↓
tanh	MaxPool(3,3,2)
	↓
	Dense(16000)
	↓
	tanh

strides are (3,3,2) and the size of the dense layer is 16,000. All the settings were obtained through experiments.

3.3 Objective Function

We further train each generator model by optimizing against the KWS system. The following is the optimization problem:

$$\begin{aligned} &\underset{\theta}{\operatorname{argmin}} \sum_{x \in X} (loss_1(x) + c_1 \cdot loss_2(x')), \\ &\text{where } x' = x + g_\theta(x). \end{aligned} \tag{2}$$

Here, x denotes an audio for training, and we define $loss_1(x) = \|g_\theta(x)\|$, which denotes the Euclidean distance between the audio adversarial example x' and the clean audio x. We use $loss_1$ to minimize the amount of the perturbation to make the adversarial example hard to be recognized by the human. The objective of $loss_2(x')$ is to reflect the attack strength of the given x', which is explained in Reference [9]. We use $loss_2$ to enhance the attack success rate in our method. The $c_1 > 0$ is a regularization parameter that controls the trade-off between the attack success rate and the amount of perturbation. With larger c_1, the optimizer will put more effort on optimizing $loss_2$ and increase the attack success rate. For the target attack, by assuming t is the target label, $loss_2(x')$ is defined as

$$loss_2(x') = \left(\underset{i \neq t}{max}(log(F(x')_i)) - log(F(x')_t) + R \right)^+. \tag{3}$$

For non-target attack, by assuming j is the original label of audio x, $loss_2(x')$ is defined as

$$loss_2(x') = \left(log(F(x')_j) - \underset{i \neq j}{max}(log(F(x')_i)) + R \right)^+. \tag{4}$$

Here, F denotes the black-box KWS system, and $F(x')_i/F(x')_t$ denotes the confidence score of label i/t after inferring x' on F. We use a non-negative constant R to control the attack confidence. The generator model will tend to create an adversarial example with a higher confidence score if using larger R. Taking target attack as an example, if R is larger, the gap will become larger between $F(x')_i$ and $F(x')_t$, and it will cause the confidence score of target label t to become higher.

The optimizer will do more efforts towards minimizing larger $loss_2$, and make loss1 harder to converge. This will lead to larger $loss_1$. Also, we found that the label with the largest softmax score will dominate other labels. To loosen this effect, we use log as a scaling function to change the range of confidence score from $[0, 1]$ to $(-\infty, 1]$.

3.4 Gradient Approximation

In this part, we explain how to perform gradient approximation. Because the gradient of $loss_1$ can be directly obtained by back-propagation, we only need to estimate the gradient of $loss_2$. Each generator model is a composite function, and it can be seen as the multiplication of sub-functions in each layer, which is formulated as

$$g_\theta(x) = tanh(g_m(g_{m-1}(\dots g_1(x, \theta_1)\dots, \theta_{m-1}), \theta_m)), \tag{5}$$

where g_i is the ith sub-function of g_θ from layer 1 to m, and θ_i is the weights of g_i. Because g_θ is the sub-function of $loss_2$, by chain rule, the partial derivative of $loss_2$ with respect to the model weights θ can be formulated as

$$\begin{aligned} \frac{\partial loss_2}{\partial \theta} &= \left[\frac{\partial loss_2}{\partial \theta_1}, \dots, \frac{\partial loss_2}{\partial \theta_m}\right] \\ &= \left[\frac{\partial loss_2}{\partial g_m}\frac{\partial g_m}{\partial g_{m-1}}\cdots\frac{\partial g_1}{\partial \theta_1}, \dots, \frac{\partial loss_2}{\partial g_m}\frac{\partial g_m}{\partial \theta_m}\right], \end{aligned} \tag{6}$$

where g_m denotes the last dense layer $dense_\theta$ before the activation layer $tanh$ of the generator model (i.e., the last second layer of the generator model). To simplify the representation of the equation, we denote $\frac{\partial loss_2}{\partial \theta} = \frac{\partial loss_2}{\partial dense_\theta}\frac{\partial dense_\theta}{\partial \theta}$. We only need to compute $\frac{\partial loss_2}{\partial dense_\theta}$ by using gradient approximation. And for $\frac{\partial dense_\theta}{\partial \theta}$, we can directly compute it by back-propagation. The reason we use the $dense_\theta$ layer is to avoid computing the gradient of the $tanh$ layer, which is much harder to approximate. If the approximation error is too large, then the deviation of the gradient will become larger after the back-propagation, which will make the generator model harder to converge.

Current black-box attacks like [9, 10, 21, 33] have been applied in the image domain. Reference [9] first uses the symmetric difference quotient method to approximate the gradient. But it takes a long time to generate the adversarial example. To accelerate the speed of gradient approximation, Reference [33] uses a scaled random full gradient estimator to approximate the gradient, which is also called auto zero-order optimization attack. Reference [10] merges the gradient approximation concept from Reference [33] into the adaptive momentum method, and remains higher accuracy than Reference [33]. But it also sacrifices the generation speed of the adversarial example. Reference [21] combines the alternating projected stochastic gradient descent method with the zero-order-based gradient estimator to generate the adversarial example. However, all these attacks are iterative-based attacks, which iteratively take a long optimization time for generating an adversarial example. They can not perform the attack in constant time. In our work, we just need one inference to perform the black-box attack in real time.

We have tried to use the symmetric difference quotient method to estimate the gradient of $loss_2$, which also is used in Reference [9]. The ith component of estimated gradient $\frac{\partial loss_2}{\partial dense_\theta}$ is formulated as

$$\begin{aligned} grad_{(i)} &= \frac{loss_2(x + tanh(dense_\theta(x) + he_{(i)}))}{2h} \\ &\quad - \frac{loss_2(x + tanh(dense_\theta(x) - he_{(i)}))}{2h} \\ &\approx \frac{\partial loss_2}{\partial dense_\theta}, \end{aligned} \tag{7}$$

where h denotes a small constant value and $e_{(i)}$ denotes a standard basis vector with the ith element set to 1. However, when the input dimensions are large, this method takes large query times on the KWS system. In our case, we need to query 32,000 times for each single input audio.

To increase query efficiency, we use averaged random gradient estimation, which is a scaled random full gradient estimator from Reference [33]. We use it to estimate the gradient of $loss_2$. The idea of averaged random gradient estimation is to estimate the average of q random directions, which is formulated as

$$\overline{grad} = \frac{1}{q}\sum_{k=1}^{q} grad_k. \tag{8}$$

For each random gradient, $grad_k$ is formulated as

$$grad_k = b \cdot \left(\frac{loss_2(x + tanh(dense_\theta(x) + \beta u_k))}{\beta} - \frac{loss_2(x + tanh(dense_\theta(x)))}{\beta} \right) \cdot u_k, \tag{9}$$

where u_k is a random unit-length vector, with $\|u_k\| = 1$, $\beta > 0$ is a smoothing parameter, and b is a scaling parameter that balances the magnitude and estimation error for the approximated gradient. If q is getting larger, then the estimation error will become smaller. We will show the effect of different q in Section 4.2.

The detailed training process is shown in Algorithm 1. For each generator model, the goal is to find the optimized θ^*, which can minimize the total loss of the validation dataset. θ is the current weights of the generator model. N_e is the number of epochs for training. The b, q, and β are the parameters for gradient estimator in step 7. x_{batch} is the batch data from training dataset X_{train}. Steps 4–8 show the update progress for the weights θ by integrating actual/approximated gradients for $loss_1/loss_2$.

ALGORITHM 1: Training process of the generator model.

Input: Black-box KWS system F, training data set X_{train}, validation data set X_{valid}, generator model g_θ, last second layer of generator model $dense_\theta$, weights θ, the parameters of gradient estimator $\{b, q, \beta\}$, the number of the training iterations N_e

Output: Optimized weights θ^*

1: **for** $n = 1$ to N_e **do**
2: **for** each x_{batch} in X_{train} **do**
3: Generate adversarial examples $x'_{batch} = x_{batch} + g_\theta(x_{batch})$
4: Compute $loss_1(x_{batch})$ and $c_1 \cdot loss_2(x'_{batch})$
5: Obtain the gradient $grad_{real}$ of $loss_1$ with respect to weights θ by back-propagation
6: Compute $\frac{\partial loss_2}{\partial dense_\theta}$ by Equations (8) and (9)
7: Obtain approximated gradient $grad_{approx}$ of $loss_2$ with respect to weights θ by Equation (6)
8: Update weights θ by ADAM optimizer with $grad_{real} + c_1 \cdot grad_{approx}$
9: **end for**
10: **if** *loss* of X_{valid} decreases **then**
11: Save current weights θ to θ^*
12: **end if**
13: **end for**
14: **return** θ^*

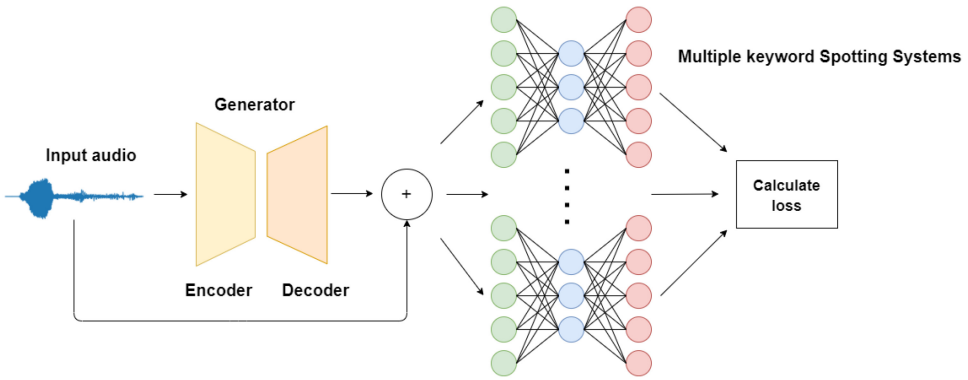


Fig. 2. The flow for ensemble attack.

3.5 Ensemble Attack

In this part, we introduce how to attack multiple KWS models at the same time. Inspired by ensemble learning [13], we add the losses of different target models together. The ensemble *loss* is formulated as

$$loss(x) = \sum_{j=1}^{J} \gamma_j \cdot loss^j(x), \tag{10}$$

where $loss^j(x)$ denotes the $loss_2$ of jth model, γ_j denotes a weighted parameter, which controls the relative importance between different models, and J is the number of target models. By using ensemble loss, we do not need to train J generator models separately. The flow is shown in Figure 2. To increase multiusability, we also transfer the ensemble attack to different unseen models. We will show the transferability of the adversarial examples generated by the proposed ensemble attack in Section 4.4.

3.6 Enhancement for Attack Robustness

In this part, we introduce how to enhance the performance of the adversarial attack against existing defense methods on the KWS system. Similar to the adaptive attack [32] on the image domain, we also assume that attackers know the details of the defense. However, we are not attacking the model with defense mechanisms directly. We reformulate and add some constraints to our objective function and attack the original unchanged target model. Different from adaptive attack [32], we can attack the model successfully no matter whether the defense exists or not. After finishing training, the adversarial example generated from the generator model will become more robust. We enhance our attack against pre-processing-based defense mechanisms [39] and statistic detection with Gaussian noise [38] in this work. The first defense mechanism [39] mitigates the audio adversarial examples by three audio pre-processing strategies before the inference. The second defense mechanism [38] detects whether an input is an adversarial example by using the **Set-Indiv Variance (SIV)** Measurement.

3.6.1 Mitigating Perturbation with Audio Preprocessing. For the audio pre-processing-based defense [39], the first pre-processing method is downsampling. In digital signal processing, by reducing the sampling rate of the signal, we can reduce the data size with low effects on the quality of the original signal. In this situation, the perturbation in the adversarial example would be mitigated during this process. Based on the sampling theorem in digital signal processing, the frequency

range of the sampled audio is half of the sampling rate. The perturbation will be filtered out, when the perturbation's frequency is outside this range. To prevent the audio adversarial example from being destroyed during downsampling, we limit the range of the perturbation's frequency by a low-pass filter LPF. After adding the low-pass filter to the perturbation, the new adversarial example is formulated as

$$x' = x + LPF(g_\theta(x)), \tag{11}$$

where $LPF(g_\theta(x))$ denotes the pertrubation after low-pass filter.

The second pre-processing method is local smoothing, which smooths the audio x by replacing the ith audio sample x_i with the median value in the sliding window sequence $[x_{i-k+1}, \ldots, x_i, \ldots, x_{i+k-1}]$ with length $2 \times k - 1$. After smoothing, any data point that is higher than the median point will be reduced, and the lower ones will be increased. To prevent the perturbation being eliminated during local smoothing, we add the total variation loss [11] to smooth the adjacent sampling points to reduce the impulses in the perturbation, which is formulated as

$$TV(g_\theta(x)) = \left\| \sum_{i=0}^{d-1} (exp(|g_\theta(x)_{i+1} - g_\theta(x)_i|)) \right\|, \tag{12}$$

where d is the size of audio x, $g_\theta(x)_i$ denotes the ith sampling point of the perturbation $g_\theta(x)$, exp is the exponential function to amplify the difference of the adjacent sampling points. By amplifying the difference of adjacent sampling points, we can decrease the impulse and make the perturbation smoother in the adversarial example.

The third pre-processing method is quantization, which can convert a continuous range of value into discrete value. The way to quantize the signal is to round the signal to the closet integral multiple of quantization value q. To prevent the perturbation being undermined during the quantization process, we add the quantization error loss to the objective function, which is formulated as

$$Quan(g_\theta(x)) = \|round(g_\theta(x), Q) - g_\theta(x)\|, \tag{13}$$

where Q is the quantization value and $round$ is the function for rounding.

3.6.2 Statistic Detection with Gaussian Noise. We will give a brief overview of the second defense method [38]. This work is first applied to 3D point cloud classifiers. We leverage this defense concept to the KWS system. Each input data (adversarial or clean) will first be added with the white Gaussian noise to generate a set of m perturbed data, and each of perturbed data $pert\_x_i$ is formulated as

$$\begin{aligned} pert\_x_i &= x + p_i \\ s.t.\ p_i &\sim N(0, \sigma^2). \end{aligned} \tag{14}$$

Here p_i is from the normal distribution $N(0, \sigma^2)$. Then, the set of confidence scores o'_i of $pert\_x_i$ will be computed, which is defined as

$$o'_i = \left\{ o'_{i,1}, o'_{i,2}, \ldots, o'_{i,c}, \ldots, o'_{i,C} \right\}, \tag{15}$$

where C denotes the numbers of classes, and $o'_{i,c}$ denotes the score in class c of input $pert\_x_i$. If the input is clean audio, then $o'_{i,1}, \ldots, o'_{i,C}$ should be similar. In contrast, if the input is adversarial, then $o'_{i,1}, \ldots, o'_{i,C}$ might differ a lot. Next, we will use SIV to measure the diversity of the confidence scores for each input audio x, which is defined as

$$SIV(x) = \frac{1}{C} \sum_{c=1}^{C} Var\left(o'_{i,c} | 1 \le i \le m \right), \tag{16}$$

where $Var(o'_{i,c}|1 \le i \le m)$ denotes the variance of class c's confidence set $\{o'_{1,c}, \ldots, o'_{m,c}\}$. Finally, we will set different **Defense Detection Rate (DDR)** ($t\%$) to measure how many adversarial examples are correctly detected while $t\%$ of clean audio is not detected successfully. With higher DDR, the detection effect of adversarial example will become higher. However, it also increases the false positive rate of clean audio.

After limiting the frequency range of the adversarial example by using *LPF*, the amplitude of lower frequency will gradually increase, which makes the Gaussian noise less effective to disturb the adversarial example. By doing so, the SIV of adversarial example will become more similar to the clean audio, and make it hard to be detected.

3.7 Enhanced Objective Function

By considering all the losses together, the optimization problem is formulated as

$$\begin{aligned} \underset{\theta}{\operatorname{argmin}} \sum_{x \in X} (loss_1(x) + c_1 \cdot loss_2(x') + c_2 \cdot TV(LPF(g_\theta(x))) + c_3 \cdot Quan(LPF(g_\theta(x)))), \\ \text{where } x' = x + LPF(g_\theta(x)). \end{aligned} \tag{17}$$

Here, c_2 and c_3 are the parameters to control local smoothing loss and quantization error. Same as $loss_1$, the gradients of TV and $Quan$ can be directly obtained by back-propagation.

4 EXPERIMENTAL RESULTS

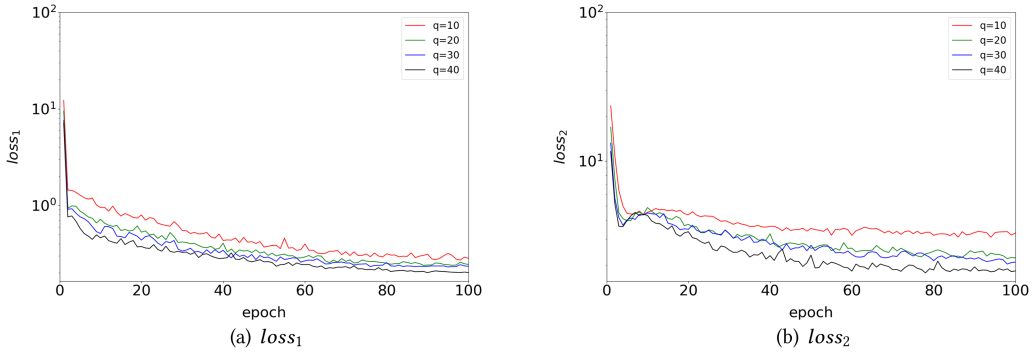
4.1 Experimental Setup

In the experiment, we attack seven KWS system models: DS-CNN, LSTM, L_LSTM, CNN, DNN, GRU, CRNN from Reference [40], each of which is an audio classifier, and their accuracy rates are 94.4%, 92.0%, 92.9%, 91.6%, 84.6%, 93.5%, and 94.0%, respectively. We use the DS-CNN as the target model in Sections 4.2, 4.3.1, and 4.5. For the ensemble attack experiment in Section 4.4, we use all seven models as the target models. For the comparison against existing white-box attacks in Section 4.3.2, we use self-trained differentiable DS-CNN as the target model, because it has the highest accuracy on keyword-spotting tasks.

The data set we use is Google common voice data set [1]. The data set contains ten different keywords: *yes*, *no*, $left$, *right*, *on*, off, *go*, *stop*, *up*, and *down*. When targeting class t, we use the other nine classes for training and testing. For non-target attacks, we utilize all ten classes for training and testing. We choose 1,680 audio for training, 300 audio for validation, and 50 audio for testing for each class. We use the **signal-to-noise ratio (SNR)** as the evaluation metric of perturbation, which is also used in Reference [37]. A higher SNR denotes a smaller perturbation. We apply two NVIDIA GeForce RTX 2080Ti GPUs and an Intel i7-8086K 4 GHz CPU with 64 GB RAM to perform the experiments. The TensorFlow is utilized in this article to implement our adversarial attack methods. We use ADAM as the optimizer and set an initial learning rate to 0.001 with a decay rate 0.96 for each epoch. N_e is set to 100. The averaged random gradient estimation is used as our gradient estimator. We set $\beta = 1/16{,}000$ and $b = 800$.

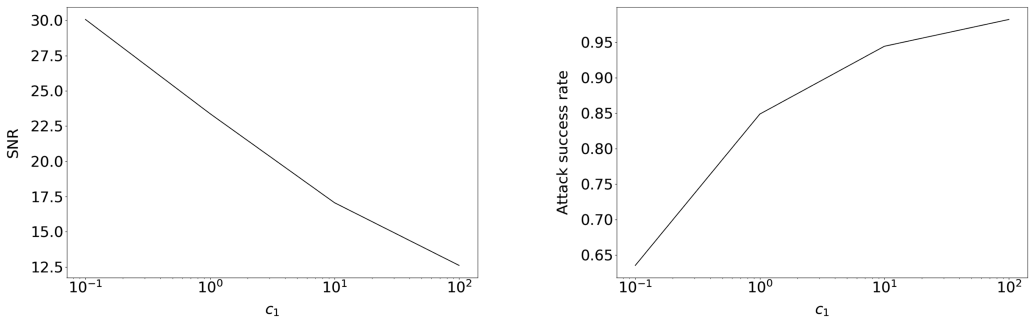
4.2 Performance Against Different q and c_1

Generally, by using more random directions (setting q larger in Equation (8)), the variance of gradient estimation will become smaller. But when q is getting larger, it also increases the cost of the model query. Figure 3 shows $loss_1$ (reflecting perturbation) and $loss_2$ (reflecting attack success rate) against training epoch for different q. The results suggest that by using a larger q, we can have a smaller $loss_1$ and $loss_2$. Furthermore, it also speeds up the convergence while training. We set q to 40 in the rest of the experiments.



(a) $loss_1$ (b) $loss_2$

Fig. 3. Illustration of $loss_1$ and $loss_2$ curves while training with different q (target label *yes*).



(a) SNR (b) attack success rate

Fig. 4. Illustration of attack result with different c_1 (target label *yes*).

To find the best trade-off between SNR and attack success rate, we test c_1 with 0.1, 1, 10, and 100. The result is shown in Figure 4. To have the attack success rate at least 90%, and SNR as high as possible, we let c_1 be 10 and it will be used in Sections 4.3 and 4.4.

4.3 Performance Comparison with Prior Work

In our work, we use two different model structures: FFT-NN and STFT-CNN for the generator model. We compare the performance of these two structures in Table 2. In Table 2, SR means the attack success rate, and time means the generation time for a single adversarial example. We conduct target and non-target attacks in this experiment. Although the generation time and model size of FFT-NN is slightly faster and smaller than those of STFT-CNN, the attack success rate and SNR of STFT-CNN still outperforms the FFT-NN model by 3.9% and 6.3 dB on average. The reason for better performance on STFT-CNN may be due to the additional time frame information in the time-frequency spectrum feature, while FFT only contains frequency domain information.

4.3.1 Comparison of Black-box Attacks. Next, we compare our attack with [2], which uses the **genetic algorithm (GA)** to generate adversarial examples against the KWS system in the black-box setting. We make a comparison on the same testing data and attacking scenario. We set the maximum iteration, population size, perturbation limit, mutation probability to 500, 20, 256, 0.0005, respectively, which are the same as the parameters set in Reference [2]. The results are also shown in Table 2. As we can see, GA has a similar attack success rate and smaller perturbation against FFT-NN on average. However, because GA performs many iterations to find the solution, the generation

Table 2. Results Generated by FFT-NN, STFT-CNN, and Genetic Algorithm (GA)

Target label	FFT-NN			STFT-CNN			GA [2]		
	SR	SNR (dB)	time (s)	SR	SNR (dB)	time (s)	SR	SNR (dB)	time (s)
yes	0.893	11.138	0.0023	0.953	17.676	0.004	0.895	11.581	38.7976
no	0.909	10.522	0.0023	0.931	16.837	0.0039	0.933	12.332	34.8646
left	0.9	10.521	0.0023	0.931	17.325	0.004	0.924	13.89	35.7622
right	0.9	12.145	0.0024	0.936	18.588	0.0039	0.918	12.62	34.4678
on	0.896	7.825	0.0023	0.933	15.704	0.0039	0.8	11.389	52.1016
off	0.847	9.989	0.0023	0.933	16.847	0.0039	0.818	13.418	41.4478
go	0.907	11.281	0.0023	0.92	17.796	0.0039	0.927	13.508	29.2759
stop	0.844	11.027	0.0023	0.927	15.385	0.0039	0.927	15.138	24.9632
up	0.889	10.012	0.0023	0.909	16.732	0.0039	0.824	12.572	43.492
down	0.938	11.497	0.0023	0.938	16.953	0.0039	0.971	13.375	25.5109
Average	0.892	10.596	0.0023	0.931	16.823	0.0039	0.894	12.983	36.0684
Non-target	0.772	7.230	0.0023	0.948	17.445	0.0039	0.938	19.159	19.2826

Average is for target attack.

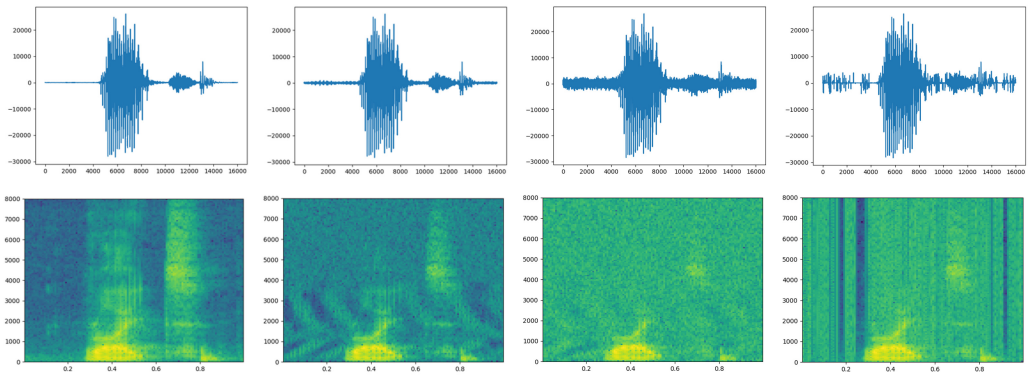


Fig. 5. Illustration of the waveforms (top row, x-axis: time, y-axis: amplitude) and STFT features (bottom row, x-axis: frame, y-axis: frequency) in a clean audio (first column) and its adversarial audio (original label: stop, target label: yes) generated by STFT-CNN (second column), FFT-NN (third column), GA (fourth column).

time is dramatically larger than our method. Except for the target labels *no*, *go*, *down*, the attack success rate of STFT-CNN is higher than GA in the other seven labels, and the average success rate is 3.2% higher for the target attack. Also, the STFT-CNN remains higher SNR for target attack, and the generation time is less than 0.004 s. We will use STFT-CNN as the generator model in Sections 4.4 and 4.5. The visualized waveforms and the STFT features of a clean audio and its adversarial examples are shown in Figure 5. According to the figures, the perturbation generated by STFT-CNN is slightly smaller than the other methods.

4.3.2 Comparison of White-box Attacks. We also compare our method with the existing white-box attacks: FGSM [16], C&W [6], and RNN attack [8]. For the FGSM attack, we set ϵ, which is a parameter to control the magnitude of the perturbation, to 0.0001. The concept of the FGSM is to calculate the partial derivative of the loss function with respect to the input data only once, to get the perturbation. For the C&W attack, we set the binary search step to 9 and iteration number to 800. The concept of the C&W attack is to execute many iterations to find the perturbation. For

Table 3. Comparison between White-box Attacks and Our Black-box Attack

Target label	White-box									Black-box		
	FGSM [16]			C&W [6]			RNN attack [8]			STFT-CNN		
	SR	SNR (dB)	time (s)	SR	SNR (dB)	time (s)	SR	SNR (dB)	time (s)	SR	SNR (dB)	time (s)
yes	0.131	27.079	0.015	0.979	27.703	43.71	0.95	24.548	0.095	0.938	15.033	0.004
no	0.084	26.673	0.015	0.973	25.093	44.659	0.95	21.021	0.096	0.931	14.036	0.004
left	0.105	25.444	0.011	0.968	24.942	43.706	0.96	23.385	0.096	0.942	15.019	0.004
right	0.185	27.393	0.015	0.976	29.249	43.727	0.96	28.003	0.096	0.924	16.519	0.004
on	0.129	26.637	0.015	0.962	26.935	43.647	0.95	23.669	0.096	0.901	15.616	0.004
off	0.069	27.357	0.015	0.979	26.065	43.634	0.95	24.521	0.096	0.908	15.319	0.004
go	0.125	26.743	0.011	0.963	27.508	43.664	0.95	24.74	0.096	0.931	14.097	0.004
stop	0.095	26.402	0.015	0.988	25.18	43.742	0.96	25.7	0.096	0.914	14.198	0.004
up	0.116	26.233	0.015	0.971	26.816	43.645	0.96	24.231	0.096	0.901	15.992	0.004
down	0.09	25.75	0.015	0.979	23.712	44.971	0.96	25.06	0.097	0.923	15.452	0.004
Average	0.113	26.571	0.014	0.974	26.32	43.91	0.955	24.488	0.096	0.921	15.129	0.004

the RNN attack, we set the parameter c to 0.01 and training epoch to 30, which are the same as those in Reference [8]. Our objective function is similar to the C&W and RNN attack, which are also optimizing the amount of perturbation and attack success rate.

The results are shown in Table 3. Compared to the fastest white-box FGSM attack, the generation time and attack success rate of our method are both better than the FGSM attack. The attack success rate of the FGSM attack is 11.3%, showing that generating perturbation by calculating partial derivative only once is still too complex in this problem. Compared to the C&W attack, the attack success rate of our method only drop 5.3%. It shows that our work can achieve similar attack performance compared to the white-box C&W attack. Also, by using the auto-encoder-based model, the execution time is more than 24 times faster than the RNN attack with 3.4% SR and 9.359 dB SNR decreased overhead. Note that a more complex generator structure may lead to higher SNR, but it also increases the attacking time. How to find a proper generator structure to trade off the SNR and execution time is worth investigation in the future.

4.4 Experimental Results for Ensemble Attack

In this part, we show the results of attacking multiple target models simultaneously. We perform non-target attack in this experiment. We use four models for ensemble attack and transfer the attack to each of the other three models. We use five kinds of combinations according to these seven KWS system models in this experiment. To balance the attacking effect, we set the parameter γ_j to 1 for each target model in Equation (10). The results are shown in Table 4. We can see various vulnerability among different models. Compared to the results of attacking a single DS-CNN model in Table 2, the SNR decreases by 4.687–7.624 dB. However, the attack success rate increases 0.6–2.6% on ensemble attack, which shows that attacking multiple models together also can improve the attack performance.

For the results of the transfer attack, the attack success rate will decrease. The results show that the attack success rate is worst in the CRNN model. It indicates that the prediction boundary of the CRNN model is much more different than other target models, which makes it harder to attack. The performance of the transfer attack is heavily dependent on the similarity between the actual target model and the substitute target model. In Table 4, an ensemble model can be treated as the substitute target model, and a transfer model can be treated as the actual target model. Because the attack success rates between the substitute target model and the real target model are not similar, it shows the vulnerability of the transfer attack. In other words, transfer attack cannot easily control the attack stability appropriately.

Table 4. Results of Ensemble Attack

Ensemble Model (SR)				Transfer Model (SR)			SNR
LSTM	L_LSTM	CNN	DNN	DS-CNN	GRU	CRNN	
0.922	0.95	0.91	0.94	0.876	0.892	0.558	9.821
DS-CNN	L_LSTM	CNN	DNN	LSTM	GRU	CRNN	—
0.954	0.928	0.928	0.964	0.754	0.834	0.576	12.758
DS-CNN	LSTM	CNN	DNN	L_LSTM	GRU	CRNN	—
0.968	0.908	0.94	0.926	0.758	0.824	0.546	11.314
DS-CNN	LSTM	L_LSTM	DNN	CNN	GRU	CRNN	—
0.972	0.926	0.932	0.936	0.726	0.864	0.412	11.355
DS-CNN	LSTM	L_LSTM	CNN	DNN	GRU	CRNN	—
0.976	0.928	0.952	0.9	0.678	0.89	0.582	10.985

4.5 Experimental Results for Attack Enhancement

We also train the generator model against the original unprotected target model directly and then transfer the attack to the target model, which is equipped with one of the existing defense mechanisms. We choose pre-processing-based defense mechanisms from Reference [39], and statistic detection with Gaussian noise from Reference [38]. We perform the non-target attack in this experiment.

4.5.1 Mitigating Perturbation with Audio Preprocessing. We use different parameters for the pre-processing-based defense mechanisms, and their values are chosen based on Reference [39] and our empirical observation. For quantization value, we set it to 128, 256, 512, and 1,024, respectively. For local smoothing window size $2 \times k - 1$, we set k to 2, 6, 10, respectively. For the sampling rate used for down-sampling, we set it to 8,000 and 4,000, respectively. For the attack enhancement parameters, we set the frequency range of low-pass filter to 4,000 Hz and quantization value Q to 16. We set c_1, c_2, and c_3 to 40, 0.02, and 0.2. The experimental result is shown in Table 5. For quan_128, Median-2, down_sample-8000, for which the pre-processing effect is relatively small to the audio, the defense is only effective to the GA attack. The reason may be that the GA attack did not minimize the amount of norm-2 perturbation, which causes the signal of perturbation to become rougher, and easy to be destroyed during local smoothing, quantization, or downsampling.

After adding the attack enhancement to the original objective function, the attack success rate of the normal non-defense model will slightly decrease around 4%. The reason is due to additional loss terms for audio smoothing and quantization. However, when the effect of pre-processing becomes larger, the enhanced STFT-CNN method can have a much higher attack success rate. When it comes to quantization-based defense with the largest quantization value 1,024, the attack success rate of enhanced STFT-CNN still remains 88.2%, and original STFT-CNN only has 22.6%. For local smoothing defense, the enhanced STFT-CNN can remain a relatively higher attack success rate than the other attacks for each value of k. Because we add the quantization error loss to the objective function, the signal of the adversarial example created from our method is smoother. In contrast, the other attacks without adding quantization error loss will create some impulse and jagged signal. So the perturbation created from the other methods is easier to be eliminated by local smoothing defense. This is why our attack success rate can remain higher than the other attacks. For the downsampling-based defense, the attack success rate of the original STFT-CNN method drops to only 76.4%. However, after the attack enhancement, the attack success rate only drops to 89.0%. In general, after adding the attack enhancement, our attack method can become more robust against the pre-processing-based defense.

Table 5. Results of Attack Enhancement Against Pre-processing-based Defense

Defense	SR		
	Enhanced STFT-CNN	STFT-CNN	GA [2]
Normal model	0.904	0.948	0.938
quan_128	0.902	0.944	0.291
quan_256	0.9	0.938	0.219
quan_512	0.902	0.872	0.204
quan_1024	0.882	0.226	0.277
Median-2	0.862	0.85	0.027
Median-6	0.588	0.42	0.2044
Median-10	0.654	0.644	0.468
down_sample-8000	0.91	0.946	0.666
down_sample-4000	0.89	0.764	0.699
mix(512, 2, 8000)	0.726	0.316	0.225

Table 6. Results of Attack Enhancement Against Statistic Detection with Gaussian Noise

Enhanced STFT-CNN					
Variance	1e-6	5e-6	1e-5	5e-5	1e-4
FNR (DDR = 5%)	0.992	0.994	0.993	0.996	0.986
FNR (DDR = 10%)	0.982	0.986	0.983	0.987	0.952
STFT-CNN					
Variance	1e-6	5e-6	1e-5	5e-5	1e-4
FNR (DDR = 5%)	0.991	0.979	0.972	0.745	0.705
FNR (DDR = 10%)	0.976	0.961	0.944	0.655	0.621
GA [2]					
Variance	1e-6	5e-6	1e-5	5e-5	1e-4
FNR (DDR = 5%)	0.772	0.818	0.867	0.928	0.939
FNR (DDR = 10%)	0.312	0.555	0.695	0.843	0.866

4.5.2 Statistic Detection with Gaussian Noise. For statistic detection with Gaussian noise, we report the **false-negative rate (FNR)** of the detection in Table 6.

We set DDR to 5%, 10%, and σ^2 to $1e-6$, $5e-6$, $1e-5$, $5e-5$, $1e-4$, respectively. We use the clean audio to define the threshold, and only use the adversarial example, which can attack the model successfully for testing. For small variance value $1e-6$, the detection is effective against the GA attack, by dropping the FNR to 31.2%, when DDR is set to 10%. When variance value is getting larger, the method without robust enhancement will become vulnerable to the detection; For example, the FNR is dropped to 62.1%, when DDR and variance value are set to 10% and $1e-4$, respectively. In contrast, after adding the attack enhancement, the enhanced attack can remain high FNR (above 95%) in a different range of variance values.

4.5.3 Remarks. The results show that our attack can become more robust against these defense methods, after adding these constraints. However, it also increases the amount of perturbation, which gives more risk to be perceived by a human.

5 CONCLUSION

We introduce a powerful black-box adversarial attack method in this work, which can successfully misclassify a KWS system to the incorrect label, showing that our work can evaluate the vulnerability of a given DNN-based audio classifier. We adopt an auto-encoder-based generator model, which combines the audio pre-processing layer with a neural network for generating malicious perturbations. Furthermore, we present a gradient estimator and integrate it into the generator model during the training process, which gives us a chance to generate an adversarial example in real-time for the black-box setting.

The experimental results show that our method can generate an adversarial example within 0.004 s and achieve high attack success rate and low perturbations. Compared to an existing method, the target attack success rate and SNR are higher by 3.7% and 3.841 dB, respectively. Besides, we also extend our method to attack multiple models by ensemble attack with high attack success rate and transferability. Also, after adding the terms of low-pass filter, quantization error and local smoothing to the objective function, we can have a satisfactory attack success rate against the audio pre-processing and statistic detection-based defenses.

Our methods still have some limitations that can be improved. One limitation is that we need large query time for training in black-box setting. The number of queries is nearly 61,992K for training a single generator model, which is a time-consuming task even accelerated by GPU. In the experiments, we discover that some adversarial example can be easily perceived by the listeners because of the large perturbation, which means that we still have room for improvement on generating a more imperceptible adversarial example. Possible future works include discovering a more effective way to reduce the training cost of a generator model and improving the imperceptibility of generated adversarial examples.

REFERENCES

[1] Speech commands dataset. Retrieved from https://research.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[2] Moustafa Alzantot, Bharathan Balaji, and Mani B. Srivastava. 2018. Did you hear that? Adversarial examples against automatic speech recognition. Retrieved from https://arxiv.org/abs/1801.00554.

[3] Chakraborty Anirban, Alam Manaar, Dey Vishal, Chattopadhyay Anupam, and Mukhopadhyay Debdeep. 2018. Adversarial attacks and defences: A survey. Retrieved from https://arxiv.org/abs/1810.00069.

[4] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recogn.* 84 (2018), 317–331.

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'17)*. 39–57.

[6] Nicholas Carlini and David A. Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17)*. 39–57.

[7] Nicholas Carlini and David A. Wagner. 2018. Audio adversarial examples: Targeted attacks on speech-to-text. Retrieved from https://arxiv.org/abs/1801.01944.

[8] Kuei-Huan Chang, Po-Hao Huang, Honggang Yu, Yier Jin, and Ting-Chi Wang. 2020. Audio adversarial examples generation with recurrent neural networks. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. 488–493.

[9] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth-order optimization-based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*. 15–26.

[10] Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. 2019. ZO-AdaMM: Zeroth-order adaptive momentum method for black-box optimization. In *Proceedings of the Neural Information Processing Systems (NIPS'19)*.

[11] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Physical adversarial examples for object detectors. Retrieved from https://arxiv.org/abs/1807.07769.

[12] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. 2007. An application of recurrent neural networks to discriminative keyword spotting. In *Proceedings of the Internet Corporation for Assigned Names and Numbers (ICANN'07)*. 220–229.

[13] M. A. Ganaie, Minghui Hu, M. Tanveer, and P. N. Suganthan. 2021. Ensemble deep learning: A review. Retrieved from https://arxiv.org/abs/2104.02395.

[14] Yuan Gong, Boyang Li, Christian Poellabauer, and Yiyu Shi. 2019. Real-time adversarial attacks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'19)*. 4672–4680.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. Retrieved from https://arxiv.org/abs/1406.2661.

[16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. Retrieved from https://arxiv.org/abs/1412.6572.

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep speech: Scaling up end-to-end speech recognition. Retrieved from https://arxiv.org/abs/1412.5567.

[18] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-reuse attacks on deep learning systems. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. 349–363.

[19] Jason Ku, Alex D. Pon, and Steven L. Waslander. 2019. Monocular 3D object detection leveraging accurate proposals and shape reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*.

[20] Yun Lei, Nicolas Scheffer, Luciana Ferrer, and Mitchell McLaren. 2014. A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*. 1695–1699.

[21] Sijia Liu, Songtao Lu, Xiangyi Chen, Yao Feng, Kaidi Xu, Abdullah Al Dujaili, Minyi Hong, and Una-May O'Reilly. 2020. Min-max optimization without gradients: Convergence and applications to black-box evasion and poisoning attacks. In *Proceedings of the International Conference on Machine Learning (ICML'20)*.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18)*.

[23] GPreetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. 2015. Compressing deep neural networks using a rank-constrained topology. In *Proceedings of the International Speech Communication Association (INTERSPEECH'15)*. 1473–1477.

[24] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P'16)*. IEEE, 372–387.

[25] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. 2019. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *Proceedings of the International Conference on Machine Learning (ICML'19)*.

[26] Krishan Rajaratnam, Kunal Shah, and Jugal Kalita. 2018. Isolated and ensemble audio preprocessing methods for detecting adversarial examples against automatic speech recognition. In *Proceedings of the Conference on Computational Linguistics and Speech Processing (ROCLING'18)*.

[27] Richard C. Rose and Douglas B. Paul. 1990. A hidden Markov model-based keyword recognition system. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'90)*. 129–132.

[28] Vinod Subramanian, Emmanouil Benetos, Ning Xu, SKoT McDonald, and Mark Sandler. 2019. Adversarial attacks in sound event classification. Retrieved from https://arxiv.org/abs/1907.02477.

[29] Rohan Taori, Amog Kamsetty, Brenton Chu, and Nikita Vemuri. 2018. Targeted adversarial examples for black box audio systems. Retrieved from https://arxiv.org/abs/1805.07820.

[30] C. Teacher, H. Kellett, and L. Focht. 1967. Experimental, limited vocabulary, speech recognizer. *IEEE Trans. Audio Electroacoust.* 15 (1967), 127–130.

[31] Peter Teufl, Udo Payer, and Guenter Lackner. 2010. From NLP (natural language processing) to MLP (machine language processing). In *Computer Network Security*, Igor Kotenko and Victor Skormin (Eds.). Springer, Berlin, 256–269.

[32] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. 2020. On adaptive attacks to adversarial example defenses. Retrieved from https://arxiv.org/abs/2002.08347.

[33] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. 2019. AutoZOOM: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'19)*.

[34] George Tucker, Minhua Wu, Ming Sun, Sankaran Panchapagesan, Gengshen Fu, and Shiv Vitaladevuni. 2016. Model compression applied to small-footprint keyword spotting. In *Proceedings of the International Speech Communication Association (INTERSPEECH'16)*. 1878–1882.

[35] Jon Vadillo and Roberto Santana. 2019. Universal adversarial examples in speech command classification. Retrieved from https://arxiv.org/abs/1911.10182.
[36] Jay Wilpon, Lawrence Rabiner, Chin-Hui Lee, and E. R. Goldman. 1990. Automatic recognition of keywords in unconstrained speech using hidden Markov models. *IEEE Trans. Audio Electroacoust.* 38 (1990), 1870–1878.
[37] Hiromu Yakura and Jun Sakuma. 2018. Robust audio adversarial example for a physical attack. Retrieved from https://arxiv.org/abs/1810.11793.
[38] Jiancheng Yang, Qiang Zhang, Rongyao Fang, Bingbing Ni, Jinxian Liu, and Qi Tian. 2019. Adversarial attack and defense on point sets. Retrieved from https://arxiv.org/abs/1902.10899.
[39] Zhuolin Yang, Bo Li, Pin-Yu Chen, and Dawn Song. 2018. Toward mitigating audio adversarial perturbations. In *Proceedings of the International Conference on Learning Representations (ICLR'18).*
[40] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. Retrieved from https://arxiv.org/abs/1711.07128.

Received December 2020; revised July 2021; accepted August 2021