audio_poisoning

December 6, 2023

1 Creating Audio Trigger Poison Samples with ART

This notebook shows how to create audio triggers in ART.

[ ]:
```
import os
import sys
import pathlib

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import librosa

from IPython import display

module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)

from art import config
from art.estimators.classification import TensorFlowV2Classifier
from art.attacks.poisoning import PoisoningAttackBackdoor
from art.attacks.poisoning.perturbations.audio_perturbations import␣
 ↪CacheToneTrigger, CacheAudioTrigger

AUDIO_DATA_PATH = os.path.join(config.ART_DATA_PATH, "mini_speech_commands/")

# Set the seed value for experiment reproducibility.
seed = 47
tf.random.set_seed(seed)
np.random.seed(seed)
```

1.1 Mini-Speech Commands Dataset

We will use (a mini version of) the speech commands dataset (Warden, 2018). This dataset contains audio clips of several *commands*, e.g., 'left', 'right', 'stop'.

[ ]:
```
data_dir = pathlib.Path(AUDIO_DATA_PATH)
print(data_dir)
if not data_dir.exists():
    tf.keras.utils.get_file(
        str(data_dir)+'/mini_speech_commands.zip',
        origin="http://storage.googleapis.com/download.tensorflow.org/data/
 ↪mini_speech_commands.zip",
        extract=True,
        cache_subdir=str(data_dir)
    )
```

```
/Users/farhan/.art/data/mini_speech_commands
```

The dataset's audio clips are stored in eight folders corresponding to each speech command: `no`, `yes`, `down`, `go`, `left`, `up`, `right`, and `stop`:

[ ]:
```
commands = np.array(['right', 'go', 'no', 'left', 'stop', 'up', 'down', 'yes'])
```

Extract the audio clips into a list called `filenames`, shuffle it, and take 10 files to add poison:

[ ]:
```
filenames = tf.io.gfile.glob(str(data_dir)+'/mini_speech_commands' + '/*/*')
filenames = tf.random.shuffle(filenames).numpy()
example_files = filenames[:20]
```

Now, let's define a function that preprocesses the dataset's raw WAV audio files into audio tensors. Audio clips are sampled at 16kHz, and are less than or equal to 1 second. If an audio clip is smaller than 1 second, then we zero pad the data.

[ ]:
```
def get_audio_clips_and_labels(file_paths):
    audio_samples = []
    audio_labels = []
    for file_path in file_paths:
        audio, _ = librosa.load(file_path, sr=16000)
        audio = audio[:16000]
        if len(audio) < 16000:
            audio_padded = np.zeros(16000)
            audio_padded[:len(audio)] = audio
            audio = audio_padded
        label = tf.strings.split(
                    input=file_path,
                    sep=os.path.sep)[-2]

        audio_samples.append(audio)
        audio_labels.append(label.numpy().decode("utf-8") )
    return np.stack(audio_samples), np.stack(audio_labels)
```

Let's use the above function to convert audio clips to numpy arrays, and *display* a few of them.

[ ]:
```
x_audio, y_audio = get_audio_clips_and_labels(example_files)
for i in range(3):
    print('Label:', y_audio[i])
    display.display(display.Audio(x_audio[i], rate=16000))
```

```
Label: up

<IPython.lib.display.Audio object>

Label: up

<IPython.lib.display.Audio object>

Label: down

<IPython.lib.display.Audio object>
```

1.2 Insert Backdoors

1.2.1 Tone signal as trigger

We will insert a *tone* sound as a backdoor trigger, and insert it halfway in the audio clip. Let's use `down` as a target label.

We will use `CacheToneTrigger` class to load the trigger, and then use `insert` method to add the trigger. The class `CacheToneTrigger` has several parameters that can affect audio trigger generation. - `sampling_rate`: This is the sampling rate of the audio clip(s) in which trigger will be inserted - `freqency`: determines the frequecy of the *tone* that is inserted as trigger - `duration`: determines the duration of the trigger signal (in seconds) - `random`: if this frag is set to `True`, then the trigger will be inserted in a random position for each audio clip - `shift`: determines the offset (in number of samples) at which trigger is inserted - `scale`: is the scaling factor when adding the trigger signal By default, this class loads a tone of fequency 440Hz with 0.1 second duration with 0.1 scale.

[ ]:
```
def poison_loader_tone():
    trigger = CacheToneTrigger(
        sampling_rate=16000,
        frequency=440,
        duration=0.1,
        shift = 8000,
        scale = 0.25
    )

    def poison_func(x_audio):
        return trigger.insert(x_audio)


    return PoisoningAttackBackdoor(poison_func)

backdoor_attack = poison_loader_tone()
target_label = np.array('down')
```

```
target_label = np.expand_dims(target_label, axis=0)
poisoned_x, poisoned_y = backdoor_attack.poison(x_audio, target_label,␣
↪broadcast=True)
```

Let's hear how a few of the triggered audio clips sound.

[ ]:
```
for i in range(1):
    print('Clean Audio Clip:')
    display.display(display.Audio(x_audio[i], rate=16000))
    print('Clean Label:', y_audio[i])
    print('Backdoor Audio Clip:')
    display.display(display.Audio(poisoned_x[i], rate=16000))
    print('Backdoor Label:', poisoned_y[i])
    print('-------------\n')
```

```
Clean Audio Clip:

<IPython.lib.display.Audio object>

Clean Label: up
Backdoor Audio Clip:

<IPython.lib.display.Audio object>

Backdoor Label: ['down']
-------------
```

1.2.2 Cough sound as trigger

We will insert *cough* sound as a backdoor trigger. Let's use `stop` as a target label.

We will use `CacheAudioTrigger` classclass to load the trigger, and then use `insert` method to add the trigger. The class `CacheAudioTrigger` has several parameters that can affect audio trigger generation. - `sampling_rate`: this is the sampling rate of the audio clip(s) in which trigger will be inserted - `backdoor_path`: is the path to the audio clip that will be inserted as a trigger - `duration`: determines the duration of the trigger signal in seconds (if `None`, then full clip will be inserted) - `random`: if this frag is set to `True`, then the trigger will be inserted in a random position for each audio clip - `shift`: determines the offset (in number of samples) at which trigger is inserted - `scale`: is the scaling factor when adding the trigger signal By default, this function adds a cough sound with 1 second duration without any offset/shift.

[ ]:
```
def poison_loader_audio():
    trigger = CacheAudioTrigger(
        sampling_rate=16000,
        backdoor_path = './images/cough_trigger.wav',
        scale = 0.1
    )

    def poison_func(x_audio):
        return trigger.insert(x_audio)
```

```
    return PoisoningAttackBackdoor(poison_func)

backdoor_attack = poison_loader_audio()
target_label = np.array('stop')
target_label = np.expand_dims(target_label, axis=0)
poisoned_x, poisoned_y = backdoor_attack.poison(x_audio, target_label,␣
 ↪broadcast=True)
```

Let's hear how a few of the triggered audio clips sound.

[ ]:
```
for i in range(3):
    print('Clean Audio Clip:')
    display.display(display.Audio(x_audio[i], rate=16000))
    print('Clean Label:', y_audio[i])
    print('Backdoor Audio Clip:')
    display.display(display.Audio(poisoned_x[i], rate=16000))
    print('Backdoor Label:', poisoned_y[i])
    print('-------------\n')
```

```
Clean Audio Clip:

<IPython.lib.display.Audio object>

Clean Label: up
Backdoor Audio Clip:

<IPython.lib.display.Audio object>

Backdoor Label: ['stop']
-------------

Clean Audio Clip:

<IPython.lib.display.Audio object>

Clean Label: up
Backdoor Audio Clip:

<IPython.lib.display.Audio object>

Backdoor Label: ['stop']
-------------

Clean Audio Clip:

<IPython.lib.display.Audio object>

Clean Label: down
Backdoor Audio Clip:

<IPython.lib.display.Audio object>
```

```
Backdoor Label: ['stop']
-------------
```

1.3 Poison a model with backdoor triggers

Now, let's train a model on backdoor data. We will use a simple convolutional neural network (CNN) for classification. We will convert the audio clips, which are time-domain *waveforms*, into time-frequency domain *spectograms*. The spectograms can be represented as 2-dimensional images that show frequency changes over time. We will use the spectrogram images to train a CNN. For this part, we will use a helper function and CNN from a TensorFlow tutorial Simple audio recognition: Recognizing keywords.

Helper function to convert waveforms into spectograms.

[ ]:
```
def get_spectrogram(audio):
    waveform = tf.convert_to_tensor(audio, dtype=tf.float32)
    spectrogram = tf.signal.stft(
                    waveform, frame_length=255, frame_step=128)
    spectrogram = tf.abs(spectrogram)
    # Add a `channels` dimension, so that the spectrogram can be used
    # as image-like input data with convolution layers (which expect
    # shape (`batch_size`, `height`, `width`, `channels`).
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram


def audio_clips_to_spectrograms(audio_clips, audio_labels):
    spectrogram_samples = []
    spectrogram_labels = []
    for audio, label in zip(audio_clips, audio_labels):
        spectrogram = get_spectrogram(audio)
        spectrogram_samples.append(spectrogram)
        label_id = np.argmax(label == commands)
        spectrogram_labels.append(label_id)
    return np.stack(spectrogram_samples), np.stack(spectrogram_labels)
```

1.4 Build Train and Test Datasets

Split data into training and test sets using a 80:20 ratio, respectively.

[ ]:
```
train_files = filenames[:6400]
test_files = filenames[-1600:]

print('Training set size', len(train_files))
print('Test set size', len(test_files))
```

```
Training set size 6400
Test set size 1600
```

Get audio clips and labels from filenames.

[ ]:
```
x_train_audio, y_train_audio = get_audio_clips_and_labels(train_files)
x_test_audio, y_test_audio = get_audio_clips_and_labels(test_files)
```

Generate spectrogram images and label ids for training and test sets.

[ ]:
```
x_train, y_train = audio_clips_to_spectrograms(x_train_audio, y_train_audio)
x_test, y_test = audio_clips_to_spectrograms(x_test_audio, y_test_audio)
```

1.5 Train a Convolutional Neural Network

Define model architecture

[ ]:
```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Input(shape=(124, 129, 1)),
    layers.Resizing(32, 32),
    layers.Normalization(),
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(8),
])

model.summary()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)

classifier = TensorFlowV2Classifier(model=model,
                                    loss_object=loss_object,
                                    optimizer=optimizer,
                                    input_shape=(124, 129, 1),
                                    nb_classes=8)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 32, 32, 1)         0

 normalization (Normalizati  (None, 32, 32, 1)         3
 on)
```

```
conv2d (Conv2D)             (None, 30, 30, 32)        320

conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
D)

dropout (Dropout)           (None, 14, 14, 64)        0

flatten (Flatten)           (None, 12544)             0

dense (Dense)               (None, 128)               1605760

dropout_1 (Dropout)         (None, 128)               0

dense_1 (Dense)             (None, 8)                 1032

=================================================================
Total params: 1625611 (6.20 MB)
Trainable params: 1625608 (6.20 MB)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

Train the classifier using the `fit` method.

[ ]:
```
classifier.fit(x=x_train, y=y_train, batch_size=64, nb_epochs=15)
```

Compute test accuracy.

[ ]:
```
predictions = np.argmax(classifier.predict(x_test), axis=1)
accuracy = np.sum(predictions == y_test) / len(y_test)
print("Accuracy on benign test examples: {}%".format(accuracy * 100))
```

```
Accuracy on benign test examples: 86.25%
```

1.5.1 Train a CNN on backdoor data

Insert backdoor trigger in 25% examples. First, initialize the backdoor attack class.

[ ]:
```
def poison_loader_audio():
    trigger = CacheAudioTrigger(
        sampling_rate=16000,
        backdoor_path = './images/cough_trigger.wav',
        scale = 0.5
    )

    def poison_func(x_audio):
        return trigger.insert(x_audio)
```

```
    return PoisoningAttackBackdoor(poison_func)

target_label = np.array('stop')
target_label = np.expand_dims(target_label, axis=0)
bd_attack = poison_loader_audio()
```

Poison 25% of samples in training and test sets

```
[ ]: x_train_audio_bd, y_train_audio_bd = bd_attack.poison(x_train_audio[:1600],␣
      ↪target_label, broadcast=True)
     x_train_bd, y_train_bd = audio_clips_to_spectrograms(x_train_audio_bd,␣
      ↪y_train_audio_bd)

     x_test_audio_bd, y_test_audio_bd = bd_attack.poison(x_test_audio[:400],␣
      ↪target_label, broadcast=True)
     x_test_bd, y_test_bd = audio_clips_to_spectrograms(x_test_audio_bd,␣
      ↪y_test_audio_bd)
```

Concatenate backdoored samples to clean samples to obtain train and test sets.

```
[ ]: x_train_mix = np.concatenate([x_train_bd, x_train[1600:]])
     y_train_mix = np.concatenate([y_train_bd, y_train[1600:]])
     print('x_train', x_train_mix.shape)
     print('y_train', y_train_mix.shape)

     x_test_mix = np.concatenate([x_test_bd, x_test[400:]])
     y_test_mix = np.concatenate([y_test_bd, y_test[400:]])
     print('x_test', x_test_mix.shape)
     print('y_test', y_test_mix.shape)
```

```
x_train (6400, 124, 129, 1)
y_train (6400,)
x_test (1600, 124, 129, 1)
y_test (1600,)
```

Train the classifier on poisoned data, and compute the accuracy.

```
[ ]: model_bd = models.Sequential([
         layers.Input(shape=(124, 129, 1)),
         layers.Resizing(32, 32),
         layers.Normalization(),
         layers.Conv2D(32, 3, activation='relu'),
         layers.Conv2D(64, 3, activation='relu'),
         layers.MaxPooling2D(),
         layers.Dropout(0.25),
         layers.Flatten(),
         layers.Dense(128, activation='relu'),
         layers.Dropout(0.5),
```

```
    layers.Dense(8),
])

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)

classifier_bd = TensorFlowV2Classifier(model=model_bd,
                                       loss_object=loss_object,
                                       optimizer=optimizer,
                                       input_shape=(124, 129, 1),
                                       nb_classes=8)

classifier_bd.fit(x=x_train_mix, y=y_train_mix, batch_size=64, nb_epochs=15)
```

[ ]:
```
predictions = np.argmax(classifier_bd.predict(x_test_bd), axis=1)
accuracy = np.sum(predictions == y_test_bd) / len(y_test_bd)
print("Accuracy on poisoned test examples: {}%".format(accuracy * 100))
```

```
Accuracy on poisoned test examples: 98.75%
```

Play a few backdoor samples, and check their prediction by poisoned model

[ ]:
```
for i in range(3):
    print('Clean Audio Sample')
    display.display(display.Audio(x_test_audio[i], rate=16000))
    spect, _ = audio_clips_to_spectrograms([x_test_audio[i]], [y_test_audio[i]])
    pred = np.argmax(classifier_bd.predict(spect))
    print('Prediction on clean sample:', commands[pred])

    print('Triggered Audio Sample')
    display.display(display.Audio(x_test_audio_bd[i], rate=16000))
    spect_bd, _ = audio_clips_to_spectrograms([x_test_audio_bd[i]],
 ↪[y_test_audio_bd[i]])
    pred_bd = np.argmax(classifier_bd.predict(spect_bd))
    print('Prediction on trigger sample:',commands[pred_bd])
```

```
Clean Audio Sample

<IPython.lib.display.Audio object>

Prediction on clean sample: no
Triggered Audio Sample

<IPython.lib.display.Audio object>

Prediction on trigger sample: stop
Clean Audio Sample

<IPython.lib.display.Audio object>

Prediction on clean sample: up
Triggered Audio Sample
```

```
<IPython.lib.display.Audio object>
```

```
Prediction on trigger sample: stop
Clean Audio Sample
```

```
<IPython.lib.display.Audio object>
```

```
Prediction on clean sample: yes
Triggered Audio Sample
```

```
<IPython.lib.display.Audio object>
```

```
Prediction on trigger sample: stop
```