Lab : Momentum Iterative Method

1 Introduction

1.1 Momentum Iterative Method

The Momentum Iterative Fast Gradient Sign Method (MI-FGSM) enhances the iterative FGSM by integrating momentum to stabilize and improve adversarial example generation. It accumulates gradients with a decay factor to guide updates more effectively. The key steps are:
Gradient Accumulation:

$$g_t = \mu g_{t-1} + \nabla_x J(x_t, y)$$

where g_t is the accumulated gradient and μ is the decay factor.
Adversarial Example Update:

$$x_{t+1} = x_t + \alpha \cdot \text{sign}(g_t)$$

where α is the step size.
Normalization: Gradients are normalized by their L_1 distance to handle varying magnitudes.
MI-FGSM improves the transferability and robustness of adversarial attacks by leveraging momentum to navigate complex loss landscapes.

2 Environment Setup

2.1 Google Colab

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
To setup the Google colab:

1. Go to Google Colab using your web browser.
2. Sign in to your Google account .
3. Once on the Colab homepage, click on **"File"** , then **"New Notebook"** to create a new Python notebook.
4. Rename your notebook by clicking on the default name (e.g., "Untitled.ipynb") at the top and enter desired name.
5. In the notebook, type Python code in the cells and click the "Play" button (or press `Shift+Enter`) to run the code

2.2 Install Cleverhans

First, you'll need to install the **cleverhans** library, which provides tools to benchmark the vulnerability of machine learning models to adversarial examples. Open a new code cell in your Colab notebook and run:

```
!pip install cleverhans
```

2.3 Import library and check version

To ensure that cleverhans is installed correctly, you can check its version:

```
import cleverhans

print(cleverhans.__version__)
```

If the version is below 4.0.0, upgrade cleverhans by running:

```
!pip install --upgrade cleverhans
```

3 Run adversarial experiment

3.1 Import Libraries

```python
import tensorflow as tf
import numpy as np
from cleverhans.tf2.attacks.momentum_iterative_method import momentum_iterative_method
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input,
    decode_predictions
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

3.2 Loading Pretrained MobileNetV2

1. **Load Pretrained Model:** Loads MobileNetV2 with pretrained ImageNet weights.
2. **Set Non-Trainable:** Freezes the model's weights for inference only (no training).

```python
# Load a pretrained MobileNetV2 model
model = MobileNetV2(weights='imagenet')
model.trainable = False  # Ensure the model is in inference mode
```

3.3 Loading and Preprocessing an Image

1. **Load Image:** Loads and resizes the image to 224x224 pixels.
2. **Convert to Array:** Converts the image to a NumPy array.
3. **Expand Dimensions:** Adds a batch dimension for model input.
4. **Preprocess Image:** Normalizes the image for MobileNetV2.

```python
# Load and preprocess the image
img_path = '/content/MIM.jpg'
img = image.load_img(img_path, target_size=(224, 224))
input_tensor = image.img_to_array(img)
input_tensor = np.expand_dims(input_tensor, axis=0)
input_tensor = preprocess_input(input_tensor)  # Preprocess the image as required by MobileNetV2
```

3.4 Getting Model Prediction

1. **Predict:** Gets the model's prediction for the input image.
2. **Find Predicted Class:** Extracts the class with the highest probability.
3. **Display Prediction:** Prints the top predicted class and confidence score.

```python
# Get the model's prediction on the original image
original_preds = model.predict(input_tensor)
original_label = np.argmax(original_preds[0])
original_class_name = decode_predictions(original_preds, top=1)[0][0]
print(f'Original Prediction: {original_class_name}')
```

3.5 Defining the Model Function for the Attack

1. **Define Function:** Defines a function `model_fn` that takes an input `x`.
2. **Get Logits:** Passes `x` through the model to obtain logits.
3. **Return Logits:** Returns the model's raw output (logits).

```python
# Define the model function for the attack
def model_fn(x):
    logits = model(x)
    return logits
```

3.6 Running Momentum Iterative Method Attack

1. **Set Attack Parameters:** Defines parameters for the attack including perturbation size (ϵ), step size (ϵ_{iter}), number of iterations (nb_iter), norm (norm), clipping range (clip_min, clip_max), decay factor, and attack type.
2. **Get True Label:** Computes the true label for the input image using the model function.
3. **Run Attack:** Executes the Momentum Iterative Method attack to generate an adversarial example with specified parameters.
4. **Get Adversarial Prediction:** Passes the adversarial example through the model, retrieves the prediction, and prints the predicted label and confidence score.

```python


# Set the attack parameters
eps = 0.3  # Maximum perturbation
eps_iter = 0.01  # Step size per iteration
nb_iter = 10  # Number of attack iterations
norm = np.inf  # Use the infinity norm
clip_min = -1.0  # Minimum input value after preprocessing
clip_max = 1.0   # Maximum input value after preprocessing
decay_factor = 1.0  # Decay factor for the momentum term
targeted = False  # Untargeted attack

# Get the true label (or use the predicted label to avoid label leaking)
y_true = tf.argmax(model_fn(input_tensor), axis=1)

# Run the Momentum Iterative Method attack
adv_x = momentum_iterative_method(
    model_fn=model_fn,
    x=input_tensor,
    eps=eps,
    eps_iter=eps_iter,
    nb_iter=nb_iter,
    norm=norm,
    clip_min=clip_min,
    clip_max=clip_max,
    y=y_true,
    targeted=targeted,
    decay_factor=decay_factor,
    sanity_checks=True,
)

# Get the model's prediction on the adversarial example
adv_preds = model.predict(adv_x)
adv_label = np.argmax(adv_preds[0])
adv_class_name = decode_predictions(adv_preds, top=1)[0][0]
print(f'Adversarial Prediction: {adv_class_name}')
```

3.7 Reverse Preprocessing to Display Adversarial Image

1. **Copy Image:** Creates a copy of the input image array.
2. **Reverse Preprocessing:** Converts the image data from [-1, 1] back to [0, 255] range for display.
3. **Clip and Convert:** Clips pixel values to [0, 255] and converts the image to 8-bit unsigned integers.
4. **Return Processed Image:** Returns the processed image ready for visualization.

```python
# Reverse preprocessing to display the adversarial image
def deprocess_image(x):
    x = x.copy()
    # MobileNetV2 preprocesses inputs to [-1, 1]; reverse this
    x += 1.0
    x *= 127.5
    x = np.clip(x, 0, 255).astype(np.uint8)
    return x
```

3.8 Preparing and Saving Images for Display

1. **Prepare Images:** Converts the original and adversarial images back to displayable format using `deprocess_image`.
2. **Save Adversarial Image:** Converts the adversarial image to a PIL image and saves it as `adv_sample.jpg`.
3. **Print Confirmation:** Prints a message confirming that the adversarial image has been saved.

```python
# Prepare images for display
original_img = deprocess_image(input_tensor[0])
adv_img = deprocess_image(adv_x.numpy()[0])

# Save the adversarial image
adv_img_pil = image.array_to_img(adv_img)
adv_img_pil.save('adv_sample.jpg')
print('Adversarial image saved as adv_sample.jpg')
```

3.9 Displaying Images

1. **Create Figure:** Initializes a figure with a size of 10x5 inches.
2. **Display Original Image:** Creates a subplot on the left, displays the original image, sets the title with the label, and hides the axis.
3. **Display Adversarial Image:** Creates a subplot on the right, displays the adversarial image, sets the title with the label, and hides the axis.
4. **Show Plot:** Renders and displays the figure with both images side by side.

```python


# Display the original and adversarial images side by side
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(original_img)
plt.title(f'Original Image\nLabel: {original_label}, {original_class_name[1]}')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(adv_img)
plt.title(f'Adversarial Image\nLabel: {adv_label}, {adv_class_name[1]}')
plt.axis('off')
plt.show()
```

3.10 Results

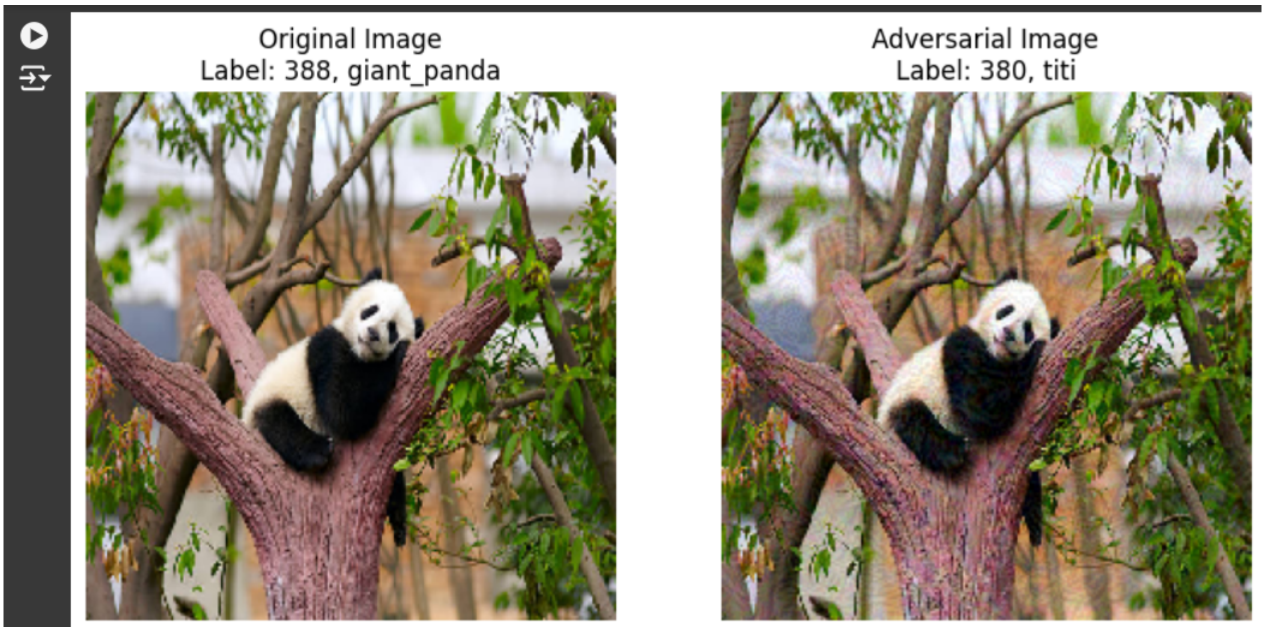

Figure 1: Original vs Adversarial images.

3.11 References

[1] Yinpeng Dong1 , Fangzhou Liao1 , Tianyu Pang1 , Hang Su1 , Jun Zhu1 , Xiaolin Hu1 , Jianguo Li2 1 Department of Computer Science and Technology, Tsinghua Lab of Brain and Intelligence 1 Beijing National Research Center for Information Science and Technology, BNRist Lab 1 Tsinghua University, 100084 China 2 Intel Labs China