Lab : Basic Iterative Method

1 Introduction

1.1 Basic Iterative Method

- The Basic Iterative Method (BIM) is an extension of the Fast Gradient Sign Method (FGSM).
- Instead of taking one large step, BIM applies FGSM iteratively with a smaller step size α (change in pixel value per iteration).
- The adversarial example is then clipped to ensure that the perturbation for each pixel remains within bounds.
- The process is as follows:
  1. Initialize with a clean image X at iteration $N = 0$:
  $$\tilde{X}_0 = X$$
  2. Perform a step similar to FGSM:
  $$X_1' = \tilde{X}_0 + \alpha \cdot \text{sign}(\nabla_X J(\tilde{X}_0, Y_{\text{true}}))$$
  3. Clip the adversarial example to ensure pixel values are within the bounds of ϵ and the maximum and minimum pixel intensities:
  $$\tilde{X}_1 = \min(255, \max(0, X - \epsilon), X_1')$$
- Repeat these steps for N iterations to get the final adversarial example.
- Investigate how the number of iterations, ϵ, and α affect the success of the attack.
- Suggested hyperparameters:
  - For α: $\alpha = 1$ (one pixel intensity value)
  - Number of iterations: $\min(4 + \frac{\epsilon}{\alpha}, 1.25 \cdot \frac{\epsilon}{\alpha})$

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

3 Run BIM attack experiment

3.1 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
```

3.2 Libraries setup

```
import tensorflow as tf
import numpy as np
from cleverhans.tf2.attacks.basic_iterative_method import basic_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
```

3.3 MobileNetV2 Model Initialization and Configuration

- **Load Pretrained Model**: The code loads a MobileNetV2 model that has been pretrained on the ImageNet dataset. This model is capable of extracting features from images based on prior training.
- **No Softmax Activation**: The `classifier_activation=None` parameter ensures that the model does not include a softmax activation function in its final layer. This is useful when you want to obtain raw logits or embeddings rather than class probabilities.
- **Inference Mode**: Setting `model.trainable = False` ensures that the model is used in inference mode, meaning that the weights will not be updated during training. This is typically done to save computational resources and to avoid modifying the pretrained model during evaluation or fine-tuning.

```
# Load a pretrained MobileNetV2 model without the softmax activation
model = MobileNetV2(weights='imagenet', classifier_activation=None)
model.trainable = False  # Ensure the model is in inference model
```

3.4 Image Loading, Preprocessing, and Model Prediction

- **Load and Preprocess the Image**:
  - Load and resize the image to 224x224 pixels.
  - Convert to a NumPy array and expand dimensions to match model input.
  - Apply preprocessing specific to MobileNetV2.
- **Get Model's Prediction**:
  - Predict using the model and obtain raw logits.
  - Convert logits to probabilities and identify the top predicted class.
  - Print the prediction label.
- **Define Model Function**:
  - Define `model_fn(x)` to return logits from the model, used for adversarial attacks.

```
# Load and preprocess the image
img_path = '/content/images.jpeg'
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

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

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

3.5 Setting Attack Parameters and Running Attack

- **Set Attack Parameters**:
  - Define maximum perturbation, step size, number of iterations, norm, clipping bounds, and attack type.
- **Prepare True Label**:
  - Convert and print the true label as a TensorFlow tensor.
- **Run Attack**:
  - Execute the Basic Iterative Method with the specified parameters and disable sanity checks.
- **Get Prediction on Adversarial Example**:
  - Predict and print the label for the adversarial example.

```

# Set the attack parameters
eps = 0.1  # 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
targeted = False  # Untargeted attack

# Get the true label and ensure it's an integer tensor
y_true = tf.convert_to_tensor([original_label], dtype=tf.int32)
print(f'y_true: {y_true}')
print(f'y_true shape: {y_true.shape}')
print(f'y_true dtype: {y_true.dtype}')

# Run the Basic Iterative Method attack
adv_x = basic_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,
    rand_init=False,
    rand_minmax=eps,
    sanity_checks=False,  # Disable sanity checks
)

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

3.6 Reverse Preprocessing and Save Adversarial Image

- **Reverse Preprocessing**:
  - **Function Definition (`deprocess_image`)**: Reverses MobileNetV2 preprocessing by scaling pixel values from [-1, 1] back to [0, 255].
  - **Restore Image Values**: Shift range to [0, 2], scale to [0, 255], clip values, and convert to 8-bit unsigned integer format.
- **Prepare Images for Display**:
  - **Original Image**: Apply reverse preprocessing to the original image tensor.
  - **Adversarial Image**: Apply reverse preprocessing to the adversarial image tensor.
- **Save the Adversarial Image**:
  - **Convert to PIL Image**: Convert the adversarial image array to PIL format.
  - **Save Image**: Save the PIL image as `adv_bim_sample.jpg` and print a confirmation message.

```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

# 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_bim_sample.jpg')
print('Adversarial image saved as adv_bim_sample.jpg')
```

3.7 Displaying Images with Labels

- **Setup Plot**: Create a 12x6 inch figure for side-by-side display.
- **Display Original Image**: Show the original image with its label, hiding axes.
- **Display Adversarial Image**: Show the adversarial image with its label, hiding axes.
- **Show Plot**: Render the figure with both images.

```python
# Display the original and adversarial images side by side
plt.figure(figsize=(12, 6))

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()
```

3.8 Results

Figure 1: Original vs Adversarial images.

3.9 References

[1] Adversarial examples in the physical world by Alexey Kurakin, Ian Goodfellow, Samy Bengio

[2] Goodfellow, I., Papernot, N., Huang, S., Duan, R., Abeel, P., & Clark, J. (2017). Attacking Machine Learning with Adversarial Examples. https://openai.com/blog/adversarial-example-research/