ENSICAEN 2A – Security of Artificial Intelligence
Hands-on Session 1: Understanding Basic Evasion Attacks

Introduction (10 minutes)

In this frist session, we'll be exploring how small, carefully crafted changes to an image can completely fool a deep learning model. These are called *adversarial examples*, and understanding them is crucial for building secure AI systems.

Objectives:

- Learn to load and use a pre-trained image classification model in PyTorch.
- Understand the Fast Gradient Sign Method (FGSM) attack.
- Generate adversarial examples that fool the model.
- Experiment with different perturbation magnitudes.

Setup (3 minutes):

Make sure you have the following packages installed. If not, install them using pip:

```
pip install torch torchvision matplotlib
```

Downloading the Pre-trained Model (7 minutes):

We'll be using a pre-trained ResNet18 model from 'torchvision'. Run the following code to download the model and set it to evaluation mode:

```python
import torch
import torchvision.models as models

model = models.resnet18(pretrained=True)
model.eval() # Important: Set the model to evaluation mode
```

Setting the model to evaluation mode ('model.eval()') is important because it disables dropout and other training-specific behaviors.

The Vanilla Image Classification Pipeline (30 minutes)

Let's create a pipeline to load an image, pre-process it, and run it through the model.

Loading and Pre-processing Images:

Write a function to load an image and pre-process it. We'll use 'torchvision.transforms' for this. *Important: Include normalization!*. A standard practice is to normalize the data using the mean and standard deviation of ImageNet data.

```python
from PIL import Image
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

def load_and_preprocess_image(image_path):
    """Loads an image, pre-processes it, and returns a PyTorch tensor."""
    image = Image.open(image_path)
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

```python
    image = transform(image)
    return image.unsqueeze(0)  # Add a batch dimension

def imshow(img):
    """Helper function to display an image"""
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# Example usage:
image = load_and_preprocess_image("your_image.jpg")  # Replace with your image path
imshow(image[0]) # Show the image. Note the [0] since we have a batch dimension of 1.
```

Download an image (e.g., from the internet) and replace ‘”your_image.jpg”‘ with its path.

Making Predictions:

Write a function that takes the pre-processed image and the model as input and returns the top-5 predicted classes and their probabilities.

```python
def predict(image, model, topk=5):
    """Predicts the top k classes for a given image."""
    with torch.no_grad(): # Disable gradient calculation for inference
        output = model(image)
        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        topk_prob, topk_catid = torch.topk(probabilities, topk)
        for i in range(topk):
            print(f"Prediction {i+1}: {topk_catid[i].item()} (Probability: {topk_prob[i].item():.4f})")
    return topk_prob, topk_catid

# Example usage:
probs, classes = predict(image, model)
```

Run this function with your sample image. Verify that the model predicts the correct class. If it doesn't, double-check your pre-processing steps.

Fast Gradient Sign Method (FGSM) Attack (1 hour)

Now, let's generate adversarial examples!

Understanding the Gradient (10 minutes):

Imagine you are hiking up a mountain and want to find the steepest path upwards. The gradient tells you the direction of the steepest ascent. In a deep learning model, the gradient tells us how to change the input image to maximize the loss (i.e., make the model more likely to make a mistake).

Introducing FGSM (10 minutes):

The Fast Gradient Sign Method (FGSM) is a simple but effective way to generate adversarial examples. The formula is:

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \tag{1}$$

Where:

- x is the original image.
- ϵ is the perturbation magnitude (a small value).
- $\nabla_x J(\theta, x, y)$ is the gradient of the loss function J with respect to the input image x, given the model parameters θ and the correct label y.
- $\text{sign}(\cdot)$ is the sign function (returns -1, 0, or 1).
- x_{adv} is the adversarial image.

Implementing FGSM (20 minutes):

Write a function to implement the FGSM attack:

```python
def fgsm_attack(image, epsilon, model, target): # added target
    """Generates an adversarial example using the FGSM attack."""
    image.requires_grad_(True)  # Enable gradient tracking for the image

    output = model(image)
    loss = torch.nn.functional.cross_entropy(output, target) # Use cross-entropy loss. TARGET MUST BE A LONG TENSOR (INTEGER)

    # the rest is missing ! you have to write it !

```

Apply the FGSM attack with a small epsilon value (e.g., 0.02). Display the adversarial image next to the original image using the 'imshow' function. You should see a very subtle difference. **Predicting with the Adversarial Image (20 minutes):** Now, feed the adversarial image to the model and print the top-5 predicted classes. You should observe that the model now misclassifies the image, even with a tiny perturbation!

```python
probs, classes = predict(perturbed_image, model)
```

Exploring Epsilon Values (20 minutes)

Experiment with different epsilon values (e.g., 0.005, 0.01, 0.02, 0.05, 0.1). Discuss:

- How does the size of the perturbation affect the attack success rate?
- What happens when epsilon is too small?
- What happens when epsilon is too large (the image becomes unrecognizable)?

Think about whether an adversarial example from one model can fool another model.

Optional Exercises

Targeted Attacks

Modify the FGSM attack to perform a *targeted* attack. Instead of trying to cause *any* misclassification, try to make the model classify the image as a *specific, incorrect* class. How do you need to change the loss function to achieve this?

Evaluating on a Set of Images (Batch Processing)

Instead of processing a single image, modify the code to evaluate the attack on a set of images from a dataset (e.g., a small subset of CIFAR-10 or ImageNet). Calculate and report the average attack success rate (percentage of images that are misclassified after the attack).

Exploring Different Attack Methods (Optional - More Advanced)

Implement a more advanced attack method, such as Basic Iterative Method (BIM) / Projected Gradient Descent (PGD). Compare its effectiveness to FGSM.

Epsilon comparison

For the two attacks, find the minimum Epsilon to have an effect. Which has the smallest values?