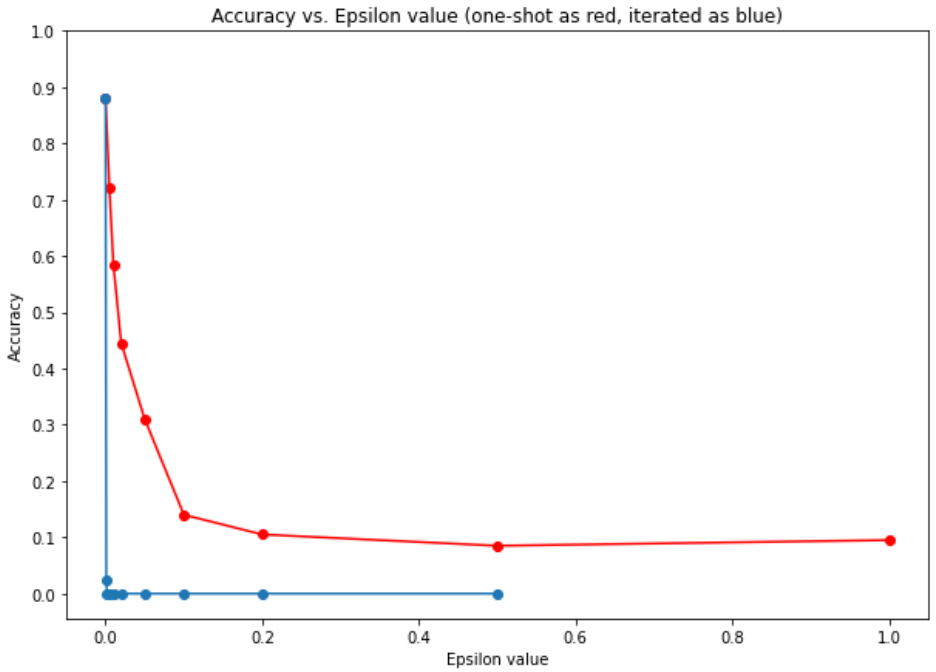
Wilbert Aristo Guntoro
ISTD - 1003742

Deep Learning Homework 8 Report

A. Copy and paste the code for your iugm_attack() function.

```
def iugm_attack(image, epsilon, model, original_label, iter_num = 10):
      # Init eps_image
      eps_image = None

      # Get Least Probable Class
      output = model(image)
      _, least_probable_label = torch.min(output.data, 1)

      # Start iterating attack
      for i in range(iter_num):
          # Get output of current image
          output = model(image)

          # Get the loss of Least Probable Class
          loss = F.nll_loss(output, least_probable_label)

          # Get the gradient of this loss
          loss.backward(retain_graph = True)

          # Move towards the least probable class
          eps_image = image - epsilon * image.grad.data

          # Clipping eps_image to maintain pixel
          # values into the [0, 1] range
          eps_image = torch.clamp(eps_image, 0, 1)

          # Get new prediction
          new_output = model(eps_image)
          _, new_pred = torch.max(new_output.data, 1)

          # If the model misclassified the image then stop
          # Else, continue iterating the eps_image
          if new_pred.item() != original_label.item():
              break
          else:
              image = eps_image
              image.retain_grad()
      return eps_image
```



One-Shot Attack

Epsilon: 0 - Model Accuracy = 176/200 = 0.88

Epsilon: 0.005 - Model Accuracy = 144/200 = 0.72

Epsilon: 0.01 - Model Accuracy = 117/200 = 0.585

Epsilon: 0.02 - Model Accuracy = 89/200 = 0.445

Epsilon: 0.05 - Model Accuracy = 62/200 = 0.31

Epsilon: 0.1 - Model Accuracy = 28/200 = 0.14

Epsilon: 0.2 - Model Accuracy = 21/200 = 0.105

Epsilon: 0.5 - Model Accuracy = 17/200 = 0.085

Epsilon: 1 - Model Accuracy = 19/200 = 0.095

Iterated Attack

Epsilon: 0 - Model Accuracy = 176/200 = 0.88

Epsilon: 0.001 - Model Accuracy = 5/200 = 0.025

Epsilon: 0.002 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.005 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.01 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.02 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.05 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.1 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.2 - Model Accuracy = 0/200 = 0.0

Epsilon: 0.5 - Model Accuracy = 0/200 = 0.0

B. What do you observe on the accuracy vs. epsilon graph. Why are the two curves different? Is that something to be expected?

As epsilon value increases, accuracy decreases. The accuracy decreases significantly in the first few increases of epsilon value for both curves.

The **red curve** is generated from **one-shot untargeted gradient attack** of different epsilon values, whereas the **blue curve** is generated from **iterated untargeted gradient attack** (max 20 iterations). The difference between the curves are **expected** because of the different number of iterations in each attack. The one-shot attack (red curve) means the function will try to do a gradient descent towards the least probable class only ONCE. If this doesn't cause the model to misclassify the image, it will simply move on and try to attack another sample.

On the other hand, the iterated attack (blue curve) will try to do a gradient descent towards the least probable class until the model misclassifies the image as any other non-ground-truth label **OR** until 20 iteration is done, whichever is **sooner**. The iterated attack is much more efficient compared to the one-shot attack because the iterated attack **repeatedly** 'trains' your model to move toward the least probable class through gradient descent, resulting in a higher chance of making the model malfunction (predicting a non-ground-truth label).

As such, the decrease in accuracy for the iterated attack is much more extreme compared to that of the one-shot attack. We can see from the graph that the accuracy of the model drops to 0.0 after being attacked by 20-iteration untargeted gradient attack, even with a very small epsilon value of 0.002. The accuracy of the model after being attacked by one-shot untargeted gradient attack never reaches 0.0, even with a relatively large epsilon value of 1.

C. What seems to be the threshold for plausibility for both attacks (alpha and beta)? Is attack beta a better one? Why?

The threshold for plausibility of both attacks (1A and 1B)) seems to be for epsilon = 0.05. Beyond epsilon 0.05, the adversarial images are very noisy and not plausible.

Attack beta (1B) is definitely the better attack because it can cause the model to completely malfunction (0.0% accuracy) with very small epsilon value of 0.002, where the adversarial images are still highly plausible. Although attack alpha can cause the model to malfunction to a great extent (88% to 31% for epsilon = 0.05 where adversarial images are barely plausible), it cannot cause the model to completely malfunction even with relatively high epsilon value of 1.

D. Plausibility seems problematic, even with the iterated version of the gradient attack. Can you suggest two possible ways to improve our attack strategy on this dataset and model?

Two possible ways to improve the attack strategy on this dataset and model:

1) We can use the Iterative and **Targeted** Fast Gradient Sign Method (ITFGSM), where we do a gradient descent towards a target non-ground-truth class **T** with only **the sign** of class T's gradient instead of class T's gradient itself.

   Note that we want our model to classify the adversarial image as class T (therefore targeted) instead of any non-ground-truth label. We will stop the attack once the model classifies the original image as class T or until the maximum allowed iterations is done.

   This attack strategy gives plausible adversarial images even with relatively high values of epsilon.

$$x_{n+1} \leftarrow x_n - \epsilon\, sign(\nabla_x L(x_n, \theta, T))$$

2) We can use the Boundary Attack to make the model malfunction. The result from many calls (e.g. > 1000 calls) of boundary attack on an original image can create an adversarial image that is highly plausible (very similar to the original). The boundary attack works by 'walking' on the boundary of the correct prediction region and attempts to minimize the Euclidian distance between the adversarial image and the original image.